MASTER EN SCIENCES INFORMATIQUES

Projet d'aide à la compréhension de l'arithmétique fondamentale

Philippe, Marc

Award date:
1994

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix

Namur

Institut d'Informatique

Projet d'aide

à la compréhension

de l'arithmétique fondamentale

Promoteur
Claude CHERTON

Mémoire présenté par
Marc PHILIPPE
en vue de l'obtention
du titre de
licencié et maître en informatique

Année académique 1993-1994

Je tiens à remercier mon promoteur, M. Cherton pour sa disponibilité et ses conseils tout au long de ce mémoire.

Ma gratitude va ensuite à M. Herman, instituteur à Natoye, pour sa précieuse collaboration à l'analyse pédagogique.

Merci enfin aux parents et amis qui ont contribué à l'élaboration de ce travail.

Résumé

Ce mémoire aborde l'étude et la réalisation d'un didacticiel pour l'arithmétique, visant à améliorer la compréhension de la notion de nombre et des quatre opérations fondamentales. L'objectif à long terme, qui dépasse le cadre de ce travail, est d'aboutir à un système d'auteur spécifique qui produise des réalisations hautement interactives.

En pratique, la matière a été restreinte aux nombres entiers, à leur addition et leur soustraction. Le produit comprend d'une part une bibliothèque d'objets programmés, d'autre part deux exercices qui l'utilisent.

Les différents chapitres présentent successivement les choix préliminaires et leur justification, l'analyse pédagogique et l'analyse informatique. La conclusion présente une évaluation de ce mémoire dans le cadre du long terme.

Abstract

In this memoire, we started on the study and implementation of a didactic programme, aimed to improve the elementary knowledge of artihmetics, e.g. the understanding of numerical systems and basic operations. Included with the long-rang forecasts - which exceed the deadline of one year - is a specific language for highly interactive applications.

Because of practical considerations, we restricted the subject matter to integer numbers, addition and substraction. The product comprises a library of programmed objects as well as two exercises using it.

The successive chapters present the justification of our preliminary choices, the pedagogical analysis and the computer processing development. By way of conclusion, we put back the achieved work in its long-term context.

Table des matières

Prologue

Aux clients qui viennent de lui acheter des légumes, le brave homme calcule le total. Quantités et prix sont notés, mais à présent le crayon parcourt le billet sans plus rien écrire. "Quelle dure journée j'ai eue !" Il vous conte des salades et son compte n'en finit pas : vous le faites pour lui.

Monsieur jardinier s'épongeant le front n'est pas une allégorie, ni un cas isolé. Au demeurant, s'il plante, c'est peut-être parce qu'il se plantait. On s'imagine l'enfant qui récitait des chiffres et des opérations comme des formules rituelles, incantations pourtant inaptes à le délivrer de son ennui et de son incompréhension. Après des années de patience, enfin autorisé à jouer dans la cour des grands, le p'tit bout d'homme aura renversé en trombe les tables de Pythagore tandis que d'autres dévastaient les jardins de l'Académie française.

Pythagore ouvrait la voie à une pensée scientifique riche. Des siècles plus tard, Archimède portait, dans un moment d'élévation, les vertus du bain au rang de principe. Et lorsque aujourd'hui encore on en tire les leçons, combien d'écoliers ne font pas la grimace avant de se plonger dans les exercices. Même revu et corrigé, le célèbre récipient a mauvaise renommée : affublée de robinets, la baignoire qui fuit est prétexte à tous les calculs.

Pour qui n'y voit goutte dès le départ, les chiffres sont une véritable conspiration. Unis comme les doigts des mains, ils se multiplient comme les lapins et divisent pour mieux régner. Ils se soustraient même à tout contrôle sûr, puisque la preuve par 9 n'en est pas une. Si ça ne colle pas, ce n'est pas juste. Mais si ça colle, demande à ton voisin, on ne sait jamais.

Les nombres sont partout dans un monde de grandes personnes. A l'heure où les enfants ne cessent de demander "pourquoi ?" et "comment ?", les adultes prennent des mesures : "quand ?", "combien ?". L'arithmétique fait partie du langage quotidien; quiconque ne la maîtrise pas se retrouve au ban quand il quitte les bancs.

Si l'instituteur veut enseigner bien, si l'élève veut bien s'efforcer, le réel décrit par des nombres pourra sembler naturel. L'espoir est là : l'arithmétique, c'est comme le chinois, ça s'apprend !

Reconnaître le terrain

Un cadre général

Le présent travail se situe à l'intersection des domaines informatique et enseignement. Il comprend l'analyse et la réalisation partielle d'un outil logiciel destiné à permettre l'apprentissage de la notion de nombre et des opérations fondamentales en arithmétique.

Le choix du sujet soulève plusieurs questions. En outre, il est de multiples façons d'aborder le problème.

Quelle place donner à l'ordinateur dans l'enseignement ?

Comment concevoir un didacticiel ?

A qui la conception pédagogique incombe-t-elle ?

Quelle orientation (architecture) choisir ?

Allons-nous étendre un système qui a fait ses preuves ou créer ex nihilo ?

Nous tenterons d'y apporter des réponses.

Vivre avec son temps

L'enseignement a la bougeotte. Il y a de la réforme dans l'air. Les enjeux d'aujourd'hui, pour faire l'objet d'un déballage public, ont sans aucun doute leur importance. Les changements dont on ne parle guère ne sont pas pour autant négligeables. Expériences pilotes et initiatives personnelles n'ont cessé de voir le jour.

Changer de pédagogie peut se faire sur trois plans :

- celui du discours, en revoyant la façon d'exposer la matière;
- celui de la méthodologie, en proposant aux élèves une organisation de travail différente;
- celui des outils. Un exemple parmi d'autres : après s'être imposée comme moyen de délassement, la télévision est aussi entrée dans les moeurs comme support à des enseignements divers (dans nos pays, la Belgique et les Pays-Bas ont pris un sérieuse avance). Le petit écran, s'il est taxé à juste titre de "chique lette de l'oeil" pour une grande part de ses émissions, mérite quelques éloges pour ses programmes éducatifs.

La révolution cathodique est terminée. Les années '90 sont celles du "bit bang". Présent dans maint foyer, le micro-ordinateur a également pris d'assaut certaines écoles. Là, parmi les utilisations de la machine, l'enseignement de la science informatique et la bureautique se

taillent la part du lion. Quid de l'ordinateur comme pur média didactique, tableau - calculateur - correcteur ?

Le logiciel éducatif aujourd'hui : forces et faiblesses

On trouve bien quelques établissements ayant tenté la grande aventure de l'ordinateur pédagogique. Dans une foule de domaines les plus variés, la micro-informatique se répand depuis un bon bout de temps. Les applications didactiques, elles, constituent le parent pauvre.

La réalisation de tout programme est soumise à une règle fondamentale : répondre aux attentes de l'utilisateur. Dans le cas qui nous occupe, le public visé est constitué par des enfants. Exploitant cette particularité au maximum, certaines maisons concoctent des applications plus attrayantes les unes que les autres. L'idée sous-jacente peut s'exprimer comme suit : donnons à l'enfant l'envie d'apprendre.

Poursuivant cette voie, des spécialistes de l'interface et de l'habillage graphiques vous produisent un petit bijou de présentation. Rien n'y manque, du dessin (qui évolue dynamiquement selon le sujet choisi ou les résultats obtenus) aux symboles en passant par un environnement plus ou moins ludique, sans oublier les animations. Cette attention est louable. Si l'on peut faire voir la matière sous un jour agréable, un grand pas est déjà franchi.

Le slogan "travailler en s'amusant" a beau nous plaire, il ne peut nous satisfaire seul. D'une part, l'élève n'est pas le seul "client". Le didacticiel est un outil parmi d'autres pour aider l'instituteur dans sa tâche. D'autre part, ce n'est pas l'existence de l'école qui fait de l'instruction un devoir. Apprendre est une nécessité, avec ou sans professeur.

Hélas, on voit des exerciseurs se contenter d'attendre les réponses pour les qualifier de bonnes ou mauvaises. Même si la façon de le dire est attrayante (un petit bonhomme qui rigole vs. une ampoule qui claque), c'est oublier la vocation première de l'application. Si l'élève vient de commettre une faute de distraction, il peut éventuellement se satisfaire d'une laconique mise en garde. Par contre, si son raisonnement est mis en cause, une machine avare en explications ne sera d'aucun secours. On ne tire les leçons de ses erreurs que si l'on en perçoit la nature.

Loin de moi l'idée qu'il n'existe pratiquement pas de didacticiels dignes de ce nom. L'E.A.O a vu s'opérer des progrès fabuleux, concrétisés par maintes applications de qualité. Malheureusement, le marché du logiciel ne reflète pas cette réalité, pour plusieurs raisons.

Le budget des écoles est rarement florissant. Or, l'achat de didacticiels passe après un investissement déjà lourd en matériel. Ensuite, ces établissements ne couvrent qu'une maigre fraction du parc de micro-ordinateurs. En somme, le marché n'est pas encore juteux.

Les entreprises qui s'y lancent toutefois doivent donc concocter des programmes bon marché. Certaines veulent produire de la qualité et ne joindront peut-être pas les deux bouts. Celles qui ne veulent pas prendre de risque ont deux solutions.

- Soit réduire les coûts. Or, un habillage graphique prend beaucoup moins de temps qu'une analyse pédagogique fouillée...
- Soit obtenir une diffusion suffisamment importante pour réduire la marge bénéficiaire par unité. Mais, il s'agit souvent de petites entreprises qui ne peuvent se permettre un marketing adapté.

Ainsi, des programmes pétris de qualités pédagogiques ne connaissent pas le retentissement qu'ils méritent. D'autant que les revues d'informatique peuvent jouer un mauvais rôle. En effet, les critiques sont souvent enthousiasmés par des logiciels ludiques qui n'apportent rien de neuf sous le soleil (ils ne remettent pas en cause la façon traditionnelle d'expliquer la matière), alors que les perles rares devraient faire la une.

Un travail d'équipe

S'il reste tant de pain sur la planche, à qui confier la mission ? Les instituteurs vont-ils s'attaquer eux-mêmes à réaliser un didacticiel ? Quelques enseignants courageux s'y sont déjà lancés. A vrai dire, forts de leur expérience pédagogique, ils connaissent mieux que quiconque les ingrédients nécessaires :

- ils savent quelle est la matière à enseigner;
- ils ont repéré les points délicats;
- ils ont mis en pratique les explications adaptées à différentes erreurs;
- ils ont imaginé des façons nouvelles de présenter la matière, ont des idées d'exercices que l'utilisation de l'ordinateur viendrait encore enrichir.

Atteindre le but passe par trois étapes : l'analyse pédagogique, l'analyse informatique et la programmation. Les deux premières sont clairement les plus difficiles.

L'analyse pédagogique

Les entraves sont de plusieurs ordres.

- Les enseignants n'ont pas tous les mêmes qualités pédagogiques ni le feu sacré.
- Tous n'ont pas la force créatrice qui leur fera imaginer une façon neuve et attrayante d'expliquer la matière.

- Il y a une marge entre l'intuition (devant une situation particulière, un instituteur trouvera le mot juste) et l'approche systématique (recenser et ramifier les erreurs possibles).

L'analyse informatique

Elle devient vite indispensable dès que l'on dépasse le cadre du mini-programme. Or elle requiert sans conteste une formation spécifique et une capacité d'abstraction élevée.

La programmation

Ici, il s'agit de choisir un langage adapté aux besoins, l'apprendre, et réaliser ses propres programmes. Toute personne ayant la moindre expérience en programmation sait le temps que nécessite une application tant soit peu fouillée et soignée, de surcroît sans bricolage ni spaghetti pour ne pas devoir réécrire à chaque fois des codes devenus hermétiques à leur propre auteur. Ne nous leurrons donc pas, l'enseignant n'aura pas assez de ses heures "perdues", à moins qu'il ne tente une reconversion complète. A ce jour, cette dernière hypothèse n'est d'ailleurs plus de la fiction...

Est-il plus facile à un informaticien de s'instituer pédagogue qu'à un enseignant de programmer ? Rien n'est moins sûr. Les deux domaines demandent des aptitudes, une formation et une expérience particulières. Entre les deux cumuls opposés, reste la solution du partage des compétences, de la mise en commun. Chacun y va des ses idées et de ses conseils, contribue à réaliser un ensemble réfléchi, cohérent, stable.

Une typologie des programmes didactiques

Produits spécifiques à l'E.A.O.

Tutoriels

Ils réalisent un enseignement programmé : la matière est présentée à l'élève sous la forme de modules restreints. Des contrôles réguliers assurent de la bonne compréhension. Selon la complexité du logiciel et la qualité des réponses fournies, la progression dans les modules sera linéaire ou ramifiée. On peut ainsi voir un tutoriel comme un manuel sur disquette.

Exerciseurs

Ces didacticiels recèlent une collection d'exercices. La machine joue le rôle d'interrogateur, corrigeant les réponses. Un diagnostic des erreurs peut être fourni.

Produits non spécifiques

Encyclopédies

Ces programmes fonctionnent comme de véritables bases de données. Les informations accessibles se composent de textes, nombres, formules, graphiques. Outre les fonctions de recherche et de restitution, on trouve aussi la possibilité de passer d'une représentation à une autre.

Au départ, les encyclopédies ne sont pas destinées à l'E.A.O., mais elles peuvent servir d'outil complémentaire.

Simulateurs

L'enseignement est un domaine d'utilisation parmis d'autres. Les simulateurs s'adressent à bien des objectifs :

- entraîner (le pilotage est l'exemple le plus évident);
- aider à la perception intuitive (lois des mouvements en physique);
- prédire en un temps raisonnable le résultat d'expériences trop lentes;
- montrer l'évolution détaillée de phénomènes trop rapides;
- évaluer la modélisation d'un phénomène trop vaste ou inexpérimentable dans la réalité;
- éviter des expériences destructrices et coûteuses.

Dans les deux premiers cas en particulier, le mode de fonctionnement se schématise comme suit : le simulateur nous plonge dans un environnement proche de la réalité. Observant le résultat de ses actions, l'élève apprend par essais et erreurs.

Types hybrides

La taxonomie définie ci-dessus n'est pas exclusive. On peut imaginer des applications qui procèdent de plusieurs types.

Simulateur + encyclopédie

La relation entre les composantes peut aller dans les deux sens.

- L'élève qui découvre la théorie est curieux de la mettre en pratique. Il peut vérifier une loi en changeant les paramètres de l'environnement.
- S'il se pose des questions, si une situation évolue d'une façon qui l'étonne, il peut en rechercher la réponse dans l'hypertexte.

Exerciseur + tutoriel

Les leçons de ce type de didacticiel pourraient faire alterner la théorie et les exerices, ces derniers servant à évaluer le niveau de compréhension atteint.

Outils de développement

Systèmes d'auteurs

Les systèmes d'auteurs sont l'oeuvre d'informaticiens qui, animés des meilleures intentions, veulent permettre aux enseignants de donner vie à leurs propres scénarios sans réinventer la roue à chaque fois.

A priori, on pourrait développer de semblables outils pour écrire n'importe quel type de didacticiel. Jusqu'à présent, les solutions envisagées conviennent pour écrire des cours programmés, mais les exercices passent souvent leur tour.

L'E.A.O. sur la sellette

Ne comparons pas l'incomparable : un ordinateur ne peut remplacer un enseignant. Même si l'affirmation paraît évidente, mettons en exergue quelques particularités.

Dans une société occidentale qui connaît une grave crise de l'éducation parentale, les instituteurs revêtent une fonction sociale de plus en plus importante. L'instruction n'est plus le seul maître mot, leur rôle d'éducateur devient primordial. Puisque le contact humain est essentiel, évitons d'isoler continuellement un enfant devant une machine.

Si, comme moi, vous refusez d'envisager une utilisation fréquente, que penser d'une utilisation parcimonieuse ? Envisageons les particularités, avantages comme inconvénients.

Indépendamment des résultats qu'il permet d'atteindre, l'ordinateur accuse au départ un désavantage sur l'être humain, en ce qu'il dispose de moins d'informations pour établir le diagnostic des erreurs. La machine peut analyser une réponse et mesurer le temps, mais elle ne voit rien de l'enfant, son visage, son regard, son attitude. Elle ne dispose pas d'un référentiel humain emmagasiné sur des décennies. Le décorticage logiciel des erreurs repose donc uniquement sur un raisonnement cartésien. De son côté, l'enseignant a comme autre outil son intuition.

Et si l'on se contentait d'un dépouillement cartésien ? L'ordinateur peut aisément dresser un historique des erreurs rencontrées, classées selon une typologie adéquate. Voilà des données pertinentes pour s'adapter au rythme et aux points faibles de l'élève. Conçue de cette manière, une leçon a sa propre dynamique. Le dialogue peut être de complexité variable, devenant plus complet, plus élaboré quand l'élève patauge.

Solution peut-être la meilleure, l'historique n'est pas pour autant une panacée universelle. En effet, le diagnostic n'est pas toujours univoque. Des raisonnements fautifs différents produiront parfois les mêmes réponses. La façon de réagir à une erreur exceptionnelle est aussi question d'appréciation. Faut-il l'imputer à de la distraction (et donc se contenter d'un avertissement) ou bien y répondre par des explications ?

Et puis le coût n'est pas négligeable... Relever toutes les causes d'erreurs possibles, organiser leur détection dans une analyse fiable, prévoir des dialogues variés. Un casse-tête en perspective pour qui souhaite répondre à toutes les attentes. Le problème se résume par cette question : comment tenir un cours particulier sans rien connaître de l'élève particulier ? Le défi est alléchant, son intérêt réside justement dans sa difficulté.

Il serait injuste de passer sous silence les domaines pour lesquels l'ordinateur est imbattable :

- l'affichage (d'une rapidité sans conteste);

- l'affichage (d'une rapidité sans conteste);
- ses possibilités graphiques (paramétrées et modulaires);
- son endurance.

Libre à nous d'utiliser PC, Mac ou autre Micro dans une présentation graphique et imagée qui allie clarté, précision, rapidité et évolution dynamique. L'avantage sur le tableau noir est ainsi patent : comparez la gestion instantanée d'un clic de la souris à l'utilisation de l'éponge et de la craie pour modifier un dessin.

Si l'on prend comme critère la richesse des situations qu'ils présentent, les scénarios de didacticiels peuvent être rangés en deux catégories:

Le scénario constant : la leçon a un contenu invariable, elle comporte toujours les mêmes valeurs, textes, graphiques, ... Inconvénient : si un module expose un problème, on n'en verra que l'instanciation prévue, sans rien pouvoir changer.

Le scénario générique : la leçon est un programme paramétrable. Un même exercice peut être fait avec des valeurs différentes. Avantage : l'élève peut envisage autant de cas qu'il le souhaite.

Qu'est-ce, une leçon idéale ? Selon moi, elle réalise trois objectifs :

- apprendre à l'enfant un certain nombre de notions;
- lui permettre d'en découvrir par lui-même;
- lui fournir des exercices au sein desquels les réponses sont analysées pour le guider dans une partie de la matière qu'il doit revoir. Des notions incomprises peuvent être revues plus lentement, et/ou avec plus de détails.

Fixer les choix

La matière

L'arithmétique est un sujet vaste, dont l'enseignement se répartit sur toutes les primaires et une partie des humanités. Vouloir le traiter intégralement serait vain et insensé. Pour aborder dans un ensemble cohérent et structuré les notions de nombre et d'opérations, il est nécessaire de restreindre le champ d'étude.

Je n'envisagerai que les entiers (ensemble Z), sans me préoccuper des nombres réels. Ces concepts nécessitent une présentation différente et des exercices particuliers.

Parmi les quatre opérations, nous pouvons distinguer d'une part, l'addition et la soustraction, d'autre part la multiplication et la division. Dans ces deux groupes, les deux opérations sont indissociables l'une de l'autre. On peut passer de l'addition à la soustraction sans inconvénient, et inversement.

Par contre, la transition d'un groupe à l'autre n'est pas négligeable. Il est nécessaire que l'élève possède bien l'addition et la soustraction avant d'aborder les deux autres opérations. En outre, la notion de partie fractionnaire est naturellement liée à la division. Evaluant le travail nécessaire pour traiter chaque opération et les contraintes de temps de ce mémoire, je m'en tiens à l'addition et à la soustraction.

En corollaire, j'exclus du cadre envisagé les concepts qui reposent sur une maîtrise des quatre opérations : le calcul des fractions et le traitement des expressions en général.

L'inspiration

A chacun son domaine, on ne s'improvise pas pédagogue. L'informaticien a beau formuler des idées et nourrir des projets didactiques, faute d'expérience sur le terrain, il peut difficilement les évaluer. Encore lui faut-il une imagination et une perspicacité peu communes pour fournir un corpus d'exercices cohérent sans avoir jamais tenu une classe.

Un instituteur a accepté de prêter sa collaboration. Marc Herman, enseignant à Natoye, se trouve être passionné par les mille et une applications que l'ordinateur pourrait trouver dans le primaire. Il fut précieux de l'entendre exposer ses méthodes et ses exercices, donner des conseils.

Dans le cadre de ce mémoire, l'échange d'idées s'est avéré une solution très réaliste pour réaliser un didacticiel.

Le type de didacticiel

A priori, le tutoriel et l'exerciseur constituent deux voies tout à fait valables. Le système d'auteur, s'il permet d'écrire des scénarios génériques, est un candidat très intéressant.

Beaucoup de logiciels d'auteurs perdent la face pour deux raisons.

1) Ils sont trop rigides. Des facilités sont offertes pour dessiner des écrans, d'accord. Mais la description des enchaînements est pauvre, le paramétrage est faible ou inexistant.

2) Ils sont trop généraux. Leur avantage est de pouvoir rédiger des leçons de n'importe quoi, disent certains. Pourtant, c'est bien là que le bât blesse. La conjugaison ne procède pas du calcul écrit, et les volumes n'ont rien à voir avec le latin. À vouloir tout permettre, on bannit ce qui est spécifique. Aucune primitive ne vous simplifiera la vie pour

- trouver la racine d'un verbe, le conjuguer;
- présenter un calcul écrit, déterminer les différents reports;
- dessiner un cône, symboliser par des segments hauteur et rayon;
- organiser un glossaire;
- ...

Do it all yourself. Pas de présentation spécifique, pas de facilités d'analyse dédicacées... Donc pas d'exercices avec correction des réponses. Résignez-vous à pondre un écran pour X=123, et un autre pour X=245.

Tutoriel, exerciseur ?

Les deux premières solutions ne sont pas exclusives. On peut très bien imaginer une application qui regroupe à la fois des exposés et des exercices. Pour avoir du succès, quelles qualités devrait avoir semblable didacticiel ? Pour commencer, des fondements pédagogiques solides. Ensuite des exercices variés. Une nécessaire cohérence dans l'organisation des modules. Sans oublier les remarques déjà faites sur le diagnostic.

Le résultat aurait l'avantage du prêt-à-l'emploi et pourrait être utilisé des années durant. Côté négatif : son caractère fermé. Il faut se contenter de ce que l'on a, on ne peut rien en jeter ni rien y ajouter. L'instituteur qui voudrait l'actualiser devrait reconstituer l'équipe pédagogue-programmeur.

Logiciel d'auteurs, langage de programmation ?

Dans le pire des cas, aucun exercice n'est fourni tout prêt, tout chaud. Mais l'enseignant peut mettre en musique bien des problèmes qui lui plaisent.

Ici aussi, la rançon du succès est élevée. Première possibilité : le langage n'est pas dédicacé à une matière spécifique. Dans ce cas, soit on choisit un bas niveau qui permet de programmer à peu près n'importe quoi à grand renfort de sport cérébral, soit on s'oriente vers un niveau plus élevé mais très figé.

Deuxième possibilité : développer un code ciblé, c.-à-d. adapté à une matière bien précise. L'avantage est clair. L'écriture de leçons et d'exercices est facilitée par des instructions riches, modulables et de haut niveau. Prenons l'exemple des conjugaisons. Imaginez un langage, associé à une petite base de données, permettant de déterminer le groupe d'un verbe et la bonne terminaison à tel mode, tel temps, telle personne.

Cette dernière solution a emporté ma préférence.

L'environnement

Parmi les micro-ordinateurs disponibles sur le marché, lequel prendre pour effectuer l'implémentation ? Si le projet a de grandes chances d'être réalisé intégralement, la question est pertinente. Hélas, dans le cadre de ce seul mémoire, on ne peut espérer obtenir une solution toute prête.

D'autres critères entrent malgré tout en ligne de compte. À performances techniques égales, le PC est moins cher que le Macintosh. Il est aussi plus répandu. Last but not least, les compatibles IBM me sont plus que familiers, contrairement aux motorolaires.

Le langage

L'analyse pédagogique de ce mémoire visait à imaginer une présentation de la matière le plus concrète possible, dans des scénarios génériques où l'élève puisse effectuer des manipulations nombreuses. Les idées et exercices envisagés ont permis de dégager un certain nombre d'éléments visuels interactifs.

- Conceptuellement, ces éléments sont des objets définis par un certain nombre de propriétes et de comportements. Leur implémentation sera facilitée dans un langage orienté objets.
- Le langage devra offrir des facilités graphiques pour représenter les objets visuels.
- Les objets virtuels correspondants devront gérer des données structurées.
- Comme les exercices conçus sont hautement interactifs, pouvoir gérer la souris est une nécessité. L'idéal est une programmation événementielle.

J'ai choisi Borland Pascal sous Windows parce qu'il répond à ces caractéristiques.

Elaborer l'outil

Faisons le point de nos résolutions. L'objectif à long terme est de réaliser un logiciel d'auteurs dédicacé à l'arithmétique.

L'idéal serait que le langage développé fournisse :

- des primitives pour le calcul décimal ou dans une base quelconque;
- des objets et éléments visuels pour la présentation de concepts;
- des facilités pour le passage d'une représentation à une autre;
- des outils pour organiser les exercices et leçons.

En conséquence, la transcription de scénarios dans ce langage serait beaucoup moins coûteuse que leur programmation à partir de rien. Est-ce à dire que le résultat présenté ici permettra à un instituteur d'écrire des programmes concis sur ordinateur aussi facilement qu'il rédige ses cours ? Non, mais ce mémoire constitue une avancée dans cette direction. Soyons terre-à-terre. Comme nous voulons obtenir "quelque chose qui tourne", espérer définir un nouveau langage et développer son compilateur (ou interpréteur) est illusoire.

Parmi les solutions réalistes, celle retenue est de construire une interface programmée en Turbo Pascal sous Windows.

Le lecteur trouvera ici les différentes étapes suivies.

1. Identifier le fond.

Cette partie du travail peut elle-même se découper en deux phases.

1. Fixer les objectifs pédagogiques, se demander quelles notions l'enfant doit acquérir.
2. Imaginer un certains nombres d'exercices qui mettent en oeuvre ces notions.

2. Produire une analyse informatique.

Les scénarios servent de base pour isoler les opérations et relever les objets qui constituteront la bibliothèque d'outils. Une fois mis en évidence, les objets sont décrits selon différents points de vue.

Leur raison d'être :

- Quelle est leur sémantique ?
- Quels concepts pédagogiques véhiculent-ils ?

Leur présence pour l'utilisateur du programme :

- Sont-ils associés à des éléments visuels ?
- Si oui, quelle représentation est utilisée ?
- Comment l'interaction avec l'utilisateur se produit-elle ?

Leur présence pour le programmeur :

- Quels services offrent-ils dans un programme ?
- Quelles sont leurs relations avec les autres objets ?
- Dans quels contextes/cadres peut-on les utiliser ?

Dans l'analyse logique, j'ai cherché à obtenir une architecture orientée objets. Comme le langage de programmation utilisé ici procède de cette philosophie, l'architecture physique ne risquait pas de s'éloigner du découpage initial. Par conséquent, le lecteur trouvera un seul texte de spécifications.

3. Passer au banc d'essais.

Les objets et traitements ainsi définis doivent encore faire leurs preuves. Pour bien montrer leur pertinence, nous les intègrerons dans deux exercices représentatifs.

Une disquette est jointe à ce mémoire. Le lecteur y trouvera le programme exécutable correspondant.

Analyse pédagogique

Objectifs

L'apprentissage d'une matière se construit comme un mur. Les notions sont apprises une par une, examinées sous tous leurs aspects. Une brique seule n'a pas valeur, elle ne prend d'importance que si elle s'agence dans un ensemble.

Une matière est bien assimilée si on la pratique sans même se rendre compte des techniques que l'on met en oeuvre. Quand les résultats sont là, nous avons souvent peine à croire que les débuts furent difficiles.

Pour être stable le mur ne doit pas seulement être bâti sans hâte ni négligence. Il doit aussi s'appuyer sur de solides fondations. En voyant l'ouvrage, personne ne songe à ses éléments invisibles mais pourtant essentiels.

Dans la matière que nous avons délimitée, le calcul écrit n'est pas le point de départ, mais bien l'aboutissement. Notre but premier n'est pas que l'élève grifonne des colonnes de chiffres comme un automate. Nous ne voulons pas qu'il applique la technique comme un truc mystérieusement juste, mais qu'il la construise lui-même en en connaissent les rouages.

La liste qui suit n'est pas exhaustive. Elle reprend un certain nombre de capacités élémentaires, une base à acquérir en priorité.

Comprendre les nombres naturels

- Dénombrer un ensemble d'objets.
- Comparer le cardinal de deux ensembles.
- Ressentir et classer différents ordres de grandeur.
- Regrouper des objets selon différents critères (bases numéraires).
- Lire / écrire le cardinal d'un ensemble d'objets[1].
- Constituer un ensemble d'un cardinal donné.
- Classer plusieurs cardinaux en ordre croissant, décroissant.

[1] Voir les détails donnés infra ("Représentation graphique")

Comprendre l'addition et la soustraction

- Réunir des ensembles d'objets (ou des ensembles disjoints, en toute généralité).
- Lier l'addition à la réunion d'ensembles.
- Prélever une partie d'un ensemble.
- Lier la soustraction à la différence entre un ensemble d'objets et une de ses parties.
- Exprimer l'opération qui correspond à une situation donnée.
- Lier la soustraction à une addition, par exemple dans une équation du type $15 + ? = 23$.
- Lier l'addition à une soustraction.
- Pour chaque forme de l'opération, retrouver l'inconnue.

Opérer sur les nombres

- Décomposer un nombre en une suite de termes.
- Rechercher des stratégies différentes pour un même calcul.
- Rechercher des opérations à résultat identique.
- Rechercher des opérations à résultats différents.
- Fixer les tables d'addition.
- Utiliser ces tables pour la soustraction.

Découvrir les propriétés

- Découvrir que l'addition est commutative, mettre en oeuvre cette propriété.
- Découvrir que la soustraction n'est pas commutative.
- Découvrir le rôle du 0 dans l'addition et la soustraction.

Public visé

Selon qu'il est dédicacé aux plus doués ou aux élèves en difficultés, un didacticiel aura une philosophie et un contenu radicalement différents.

Les élèves doués pourraient s'en sortir seuls pour une grande partie de la matière. Expliquez-leur le système décimal et ils sont partis. Ce qu'est un nombre, ils le savent intuitivement, et le système décimal n'est après tout qu'une convention parmi d'autres, dont le grand mérite est de répondre à une logique simple qu'ils maîtrisent d'emblée. Le schéma d'un calcul écrit ne leur pose pas vraiment de problème.

À l'opposé, certains de leurs copains de classe écarquilleront les yeux dès le départ. Rien que l'écriture d'un cardinal en chiffres posera souvent des problèmes. Parfois c'est la compréhension du système décimal qui est en cause, parfois ils ont même du mal à se représenter les différents ordres de grandeur. Ici, pas question de partir de l'écriture abstraite. Les explications données devront être le plus concrètes possible, il faudra multiplier les échelons qui conduisent de la base au sommet de la matière.

Dans le premier cas, un logiciel de drill fera l'affaire et conduira les bons éléments à leur pleine réussite. Certes, ils y ont droit. Mais si je consacrais ce mémoire à leur concervoir un didacticiel, je ne penserais pas avoir fait oeuvre utile. Car les exercices de drill sont relativement faciles à écrire, ne demandent pas d'interface très élaborée ni d'explications fouillées.

Par contre, les élèves en difficulté trouveraient une aide appréciable si un logiciel leur expliquait la notion même de nombre. Sur un plan plus philosophique, le but n'est pas de leur fournir du bois de rallonge (il suffirait de continuer à leur inculquer des trucs) mais de faire tomber la pièce, pour qu'ils aient confiance en eux-mêmes et utilisent leurs propres ressources pour réussir comme les autres.

Représentation graphique

Mais comment représenter concrètement un nombre ? Quelle convention utiliser qui facilite le passage au système décimal ? Il est intéressant de montrer à l'élève que le choix d'un système numérique est arbitraire mais important. Idéalement, le passage du nombre lui-même à sa représentation décimale s'opère en trois étapes.

1) Un nombre cardinal est représenté par un ensemble d'objets. Pour une quantité mesurable non dénombrable, un liquide ou du sable sert de support matériel.

L'enfant aime manipuler et prend une part active dans la leçon. L'aspect ludique de son activité suscite sa motivation.

Par l'expérience pratique, l'élève se rend compte que le nombre 17 est bien concret.

Il utilise la vue, le toucher et ses mains pour se forger une image mentale d'une quantité.

2) L'élève passe à une représentation symbolique.

A priori, chaque convention est équivalente. Le système unaire[2] est conceptuellement le plus simple et possède deux variantes. La première : le dessin d'une pomme figure la pomme elle-même; la correspondance entre l'objet et l'idée intuitive est immédiate. La seconde : un point (ou un trait) vaut pour un objet quelconque, c'est un pas supplémentaire vers l'abstraction. Pour un ensemble d'objets, le symbole choisi est répété autant de fois que nécessaire.

L'inconvénient de ce système saute aux yeux quand l'enfant représente des gros ensembles d'objets. Il commence par voir que l'écriture et la lecture d'un nombre sont longues. Ensuite, il ne peut comparer avec précision deux grandes quantités.

Il se tire d'embarras par des regroupements réguliers. D'abord à un niveau, ensuite pour plusieurs ordres de grandeur. Reste à élaborer en conséquence de nouveaux idéogrammes ou pictogrammes, selon la direction choisie au départ.

3) Les chiffres servent d'alphabet numérique standard.

L'instituteur montre comment perfectionner le système pour qu'il soit encore plus rapide et plus pratique. D'une part, l'écriture remplace le dessin. D'autre part, les chiffres 1 à 9 sont utilisés pour tous les ordres de grandeur. La position détermine l'importance, ce qui amène l'usage du 0 pour "combler les trous".

Pour ce mémoire, il me semble judicieux d'offrir des objets programmés qui correspondent aux systèmes des deux dernières étapes.

---

[2]Le nom ne désigne pas un système *positionnel*, comme le décimal ou le binaire (car alors on ne pourrait représenter en unaire que la valeur nulle), mais un système *d'énumération*. En pratique, l'écriture d'un entien *n* en unaire comporte *n* symboles.

Pour la représentation symbolique en particulier, je choisis d'utiliser les pictogrammes suivants :

Image	*Nom utilisé*	*Valeur*
	bonbon	unité
	sachet	B bonbons
	caisse	B sachets
	armoire	B caisses

où B représente la base numérique utilisée. Ces symboles valent bien entendu pour les nombres naturels.

Comment afficher un nombre négatif dans ce système ? Pour rester cohérent, il faut bien reprendre les mêmes dessins. L'inversion des couleurs est un signe distinctif possible. Ainsi, chaque objet appraît en négatif comme ci-dessous :

Pour montrer les regroupements et introduire le concept de base, deux ordres de grandeur sont un minimum. Trois rangs permettent la généralisation et peuvent suffire. J'ai toutefois choisi d'en présenter quatre, pour donner une marge supplémentaire. Libre à l'utilisateur ou au rédacteur des exercices de limiter la taille des nombres.

Dans la suite de ce mémoire, je désignerai par *pictogramme canonique* d'un nombre l'affichage d'une valeur en icônes avec les conditions suivantes :

- l'ensemble comprend des icônes d'un seul signe;
- si B désigne la base, l'ensemble a au maximum B-1 icônes dans chaque ordre de grandeur.

L'ensemble constitué seulement de bonbons sera appelé *pictogramme unaire*.

Les termes génériques (sans distinction entre les ensembles canoniques, non canoniques et unaires) seront *nombre en bonbons*, *nombre en icônes* ou encore *pictogramme d'un nombre*.

Scénarios

En exploitant l'idée exposée ci-dessus, on peut imaginer une foule de scénarios possibles. En voici quelques-uns.

1. Les ordres de grandeur

Au début, l'enfant manipule de petites quantités de bonbons virtuels (pictogrammes unaires). On lui demande des les compter, ce qu'il fait sans peine et sans caries. Dans un second temps, il est confronté à une grande quantité de bonbons, cinquante ou cent. Ici, à défaut de connaître l'astuce, enfants comme adultes se trompent facilement.

La planche de salut, ce sont les regroupements. Chaque fois que je sélectionne dix bonbons, pof, je vois apparaître à leur place un sachet. Si nécessaire, des sachets cèderont la place à une caisse. Maintenant, un coup d'oeil permet de saisir la quantité totale.

Cet exercice montre à l'enfant les deux avantages d'un tel système numérique.

1. Avec plusieurs ordres de grandeur, des cardinaux importants se manipulent plus aisément.
2. La lecture et l'écriture de nombres sont plus rapides.

2. Les comparaisons d'ensembles

Dans un premier temps, l'élève ne connaît que les pictogrammes de nombres. Déjà à ce stade, il est indispensable qu'il puisse comparer des quantités. L'ordinateur présente deux valeurs encadrées, par exemple deux sachets et trois bonbons à gauche, trois sachets et deux bonbons à droite. Cela fait le même total d'objets dans les deux cas. Si l'élève ne saisit pas que le deuxième ensemble est plus important, il peut déballer les sachets pour n'avoir que des bonbons dans les deux cadres.

Notons que même pour ce stade élémentaire, une gradation des difficultés est bienvenue. Pour bien fixer les ordres de grandeur, mieux vaut commencer par des ensembles qui comportent chacun un seul ordre de grandeur, comme x caisses à gauche, y sachets à droite. Ensuite seulement viennent les représentations de nombres à deux, puis trois chiffres significatifs.

Une fois capable de comparer deux nombres en icônes, l'enfant peut classer plusieurs ensembles, en ordre croissant ou décroissant.

3. Le lien entre icônes et chiffres

L'ordinateur choisit un nombre et l'affiche (en chiffres). À partir d'un stock d'objets, l'enfant doit constituer la représentation pictographique correspondante.

Le programme affiche un nombre en icônes et propose plusieurs valeurs en chiffres. L'élève doit choisir celle qui correspond. Plus tard, l'élève ne choisit plus parmi plusieurs propositions mais écrit lui-même le nombre.

Ces exercices sont primordiaux pour trois raisons.

1. L'élève consolide la compréhension des ordres de grandeur. Il voit ce que désignent 2, 20, 200.
2. Il comprend que l'ordre des chiffres est important. L'exemple suivant parle de lui-même. L'ordinateur propose comme ensemble : quatre caisses et cinq sachets. Les trois nombres affichés sont 450, 45 et 540.
3. Il saisit l'importance du zéro à l'intérieur d'un nombre. Deux caisses et trois bonbons, ça fait 23 ou 203 ?

4. La comparaison de nombres

Le principe est le même qu'au point deux. Si l'enfant se trompe, le programme lui affiche les représenations en bonbons et le laisse travailler comme précédemment.

5. L'addition en bonbons

L'ordinateur présente deux ensembles canoniques d'objets. L'élève les remet ensemble dans un même cadre. À ce stade, il comprend ce que signifie additionner.

Mais peut-il écrire le résultat en chiffres ? S'il y a quatre sachets et douze bonbons, pas question qu'il écrive 412. Pour trouver la bonne solution, il faut effectuer un regroupement.

Dès lors, les points acquis sont les suivants :

- L'écriture d'un nombre suit une règle fondamentale : il y a un chiffre pour chaque rang, un rang pour chaque chiffre.
- Entre un ensemble de pictogrammes et l'écriture en chiffres du cardinal, il peut y avoir plusieurs transformation à effectuer : d'une part des regroupement, d'autre part un tri des ordres de grandeur.
- Le choix de la base déterminie un chiffre maximum à ne pas dépasser.

- Une fois les regroupements assimilés, la notion de report dans un calcul écrit ne sera pas étrangère.

6. La soustraction en bonbons

Le déroulement sera similaire à celui de l'addition.

Pour l'addition, le principe fondamental est le regroupement. Le cardinal d'une réunion d'ensembles positifs ne peut s'écrire directement en chiffres s'il y a plus de B objets d'un ordre de grandeur quelconque, B figurant la base numérique.

Ici, la notion à assimiler est l'ouverture. Comme on ne peut écrire de nombre avec des chiffres positifs et négatifs, il faudra effectuer deux types d'actions.

- L'effacement : six bonbons positifs et six bonbons négatifs sélectionnés s'annulent.
- L'ouverture : s'il ne reste que des bonbons négatifs, il faut ouvrir un sachet positif et reprendre l'effacement.

Analyse logicielle

Introduction

L'écriture du code BPW (Borland Pascal for Windows) - et en particulier le choix des identificateurs - répond à quelques conventions personnelles que je présente ici.

- Le nom d'un type structuré (RECORD ou OBJECT) commence par la lettre T.
- Le nom d'un type ordinal (entier ou énuméré) commence par V.
- Le nom d'un type pointeur commence par P. Si le type pointé correspondant a le préfixe T, ce T est omis pour le pointeur.
- Ce qui suit le préfixe T, V ou P d'un type est contruit sur des mots ou abréviations de mots français.
- Si la première lettre d'une variable est T, V ou P, elle est initiale d'un mot français ou de son abréviation.
- Tous les types d'objets descendant de TBoite ont un nom qui commence soit par TB, soit par TBoite.
- Le préfixe TBoite est destiné aux objets intermédiaires, non utilisables tels quels dans un exercice. Exemple : TBoiteCapture.
- Le préfixe TB est destiné aux objets dont le comportement a été complètement défini pour une utilisation particulière dans un exercice. Exemple : TBSimpleAdd, descendant de TBoiteCapture.
- Les instances de ces types commencent par B.

La description de chaque objet suivra la même découpe.

1. "Sémantique" : cadre d'utilisation et le comportement de l'objet.
2. "Champs" : identificateur, type Pascal et sémantique de chaque donnée.
3. "Méthodes" : pour chaque procédure et fonction, je donnerai sa raison d'être, puis les paramètres en entrée et en sortie (selon le même canevas que pour les champs).

Pour information, un astérisque * placé en regard d'un scalaire, tableau ou enregistrement désigne un type intermédiaire qui n'appartient ni au Pascal standard, ni à Windows. Le lecteur en trouvera la déclaration Pascal dans l'*Unit Globaux*, fournie en annexe.

Objets associés à un élément visuel

TBoîte

Sémantique

Cet objet a pour buts de :

- gérer un élément visuel de forme rectangulaire, qui présente un contour, une barre de titre et une barre de sous-titre (facultative);
- servir de structure générique à tous les objets qui gèrent semblable cadre (cf. infra, "Hiérarchie des objets").

Champs

Identificateur	*Type*	*Sémantique*
X1, Y1	Integer	Coordonnée du coin supérieur gauche de la boîte
X2, Y2	Integer	Coordonnée du coin inférieur droit de la boîte
Fenêtre	PWindow	Adresse de la fenêtre qui contient la boîte
Peinte	Boolean	Indique si la boîte est affichée
Active	Boolean	Indique si la boîte peut réagir aux clicks de la souris
Bord	TPoint*	Marge à respecter entre le contour et le contenu affiché
Titre	PChar	Chaîne de caractères affichée dans la barre de titre
SousTitre	PChar	Chaîne de caractères affichée dans la barre de sous-titre
OKSousTitre	Boolean	Indique si le sous-titre doit être affiché
EncreTitre	TColorRef	Couleur d'encre du titre et du sous-titre
FondTitre	TColorRef	Couleur de fond du titre et du sous-titre
Plume	HPen	Plume pour tracer les traits de la boîte
Effaceur	HPen	Plume pour effacer les traits de la boîte
BrosseFond	HBrush	Brosse pour effacer l'intérieur de la boîte
BrosseTitre	HBrush	Brosse qui peint les barres de titre
FlButton	VFlButton*	Utilisation actuelle du bouton gauche de la souris dans la boîte

Méthodes

constructor Init

Objet : initialiser les champs de l'objet

Entrée :

UnParent	PWindow	Adresse de la fenêtre qui contient la boîte
UnTitre	PChar	Chaîne de caractères du titre
TitreBis	Boolean	Indique s'il faut créer une barre de sous-titre
AX, AY, BX, BY	Integer	Coordonnées des coins supérieur gauche et inférieur droit de la boîte

destructor Done

Objet : détruire les variables dynamiques qui dépendent de l'objet TBoîte (plumes, brosses, chaînes de caractères, ...)

procedure AfficheContenu

Objet : afficher l'intérieur de la boîte (comme TBoîte est un objet générique sans contenu, cette méthode redessine seulement le fond blanc).

Entrée :

DC	HDC	Handle d'un Display Context de la fenêtre qui contient la boîte

procedure AfficheSousTitre

Objet : dessiner la barre de sous-titre et son texte.

Entrée :

DC	HDC	Handle d'un Display Context de la fenêtre

Précondition :

DC désigne un Display Context valide.

Cette précondition est valable pour toutes les méthodes (de cet objet et de ses descendants) qui reçoivent un Display Context en paramètre, à l'exception de la procédure PrendDC.

procedure AfficheTitre

Objet : afficher la barre de titre et son texte

Entrée :

DC	HDC	Handle d'un Display Context de la fenêtre

`procedure Cache`

Objet : effacer toute la boîte (intérieur, barres et contours).

Entrée :

DC	HDC	Handle d'un Display Context de la fenêtre

`function Contient`

Objet : déterminer si un point de la fenêtre appartient à la surface de la boîte.

Entrée :

Msg	TMessage	Paramètres d'un message de Windows, comprenant entre autres les coordonnées relatives du pointeur de la souris dans la fenêtre

Sortie : VRAI ssi le point (Msg.lParamLo, Msg.lParamHi) est inscrit dans les contours de la boîte.

`procedure EffaceContenu`

Objet : effacer l'intérieur de la boîte.

Entrée :

DC	HDC	Handle d'un Display Context de la fenêtre

`procedure LacheDC`

Objet : désallouer un Display Context, dont le handle est remis à 0.

Entrée/Sortie :

DC	HDC	Handle d'un Display Context

procedure Montre

Objet : afficher toute la boîte (contours, barres, intérieur).

Entrée :

DC	HDC	Display Context valable pour la fenêtre
Info	TPaintStruct	Contient entre autres les coordonnées des régions de la fenêtre à redessiner

procedure PrendDC

Objet : allouer un display context valable pour la fenêtre de la boîte.

Sortie :

DC	HDC	Handle d'un Display Context

procedure PrendSousTitre

Objet : changer et afficher le texte du sous-titre.

Entrée :

UnSousTitre	PChar	Texte du nouveau sous-titre

procedure lButtonDown

Objet : répondre à un click du bouton gauche de la souris dans la boîte.

Entrée :

Msg	TMessage	Paramètres d'un message wm_lButtonDown

procedure lButtonDblClk

Objet : répondre à un double click du bouton gauche de la souris dans la boîte.

Entrée :

Msg	TMessage	Paramètres d'un message wm_lButtonDblClk

procedure lButtonUp

Objet : répondre au relâchement du bouton gauche de la souris, soit parce que le pointeur se trouve dans la boîte à ce moment précis, soit parce que le pointeur se trouvait dans la boîte quand le bouton a été enfoncé.

Entrée :

Msg	TMessage	Paramètres d'un message wm_lButtonUp

procedure lButtonShift

Objet : répondre à un click du bouton gauche alors que la touche Shift est enfoncée.

Entrée :

Msg	TMessage	Paramètres d'un message wm_lButtonDown

procedure MouseMove

Objet : répondre au déplacement du pointeur dans la boîte.

Entrée :

Msg	TMessage	Paramètres d'un message wm_MouseMove

procedure rButtonDown

Objet : répondre à un click du bouton droit de la souris.

Entrée :

Msg	TMessage	Paramètres d'un message wm_rButtonDown

procedure rButtonUp

Objet : répondre au relâchement du bouton droit de la souris.

Entrée :

Msg	TMessage	Pramètres d'un message wm_rButtonUp

TBTexte

Sémantique

TBoîte sans sous-titre qui permet l'affichage d'un texte. Le click d'un quelconque bouton de la souris est sans effet.

Champs

Identificateur	*Type*	*Sémantique*
EncreTexte	TColorRef	Couleur d'encre du texte affiché comme contenu
Texte	Array[] of Char	Chaîne de caractères servant de contenu

Méthodes

`procedure AfficheContenu`

Objet : afficher *Texte* à l'intérieur de la boîte

Entrée :

DC	HDC	Display Context valable pour la fenêtre
Info	TPaintStruct	Désigne entre autres la région de la fenêtre à repeindre

`procedure PrendTexte`

Objet : changer et afficher le texte de la boîte.

Entrée :

UnTexte	TCommentaire*	Chaîne de caractères à afficher

`procedure VideContenu`

Objet : effacer et détruire le texte de la boîte.

TBoîteCases

Sémantique

TBoîte dont le contenu est divisé en cases identiques. Les cases sont numérotées par un singleton, ligne par ligne et de gauche à droite. La case supérieure gauche porte le numéro 0. Notons que seule la surface à l'intérieur des bords est divisée (c.-à-d. la boîte sans les barres de titre ni les marges). TBoîteCases n'a pas d'utilisation directe mais sert d'objet générique pour des applications spécifiques.

Champs

Identificateur	*Type*	*Sémantique*
DimCase	TPoint*	Largeur et hauteur d'une case (en pixels)
NbCases	TPoint*	Nombre de cases dans la boîte, en largeur et en hauteur
PlumeCase	HPen	Plume pour dessiner le contour d'une case
Retrait	TRectangle*	Marge à respecter dans chaque case pour y afficher son contenu

Méthodes

`constructor Init`

Objet : Initialiser les champs de l'objet

Entrée :

UnParent	PWindow	Adresse de la fenêtre qui contient la boîte
UnTitre	PChar	Titre de la boîte
TitreBis	Boolean	Indique s'il y aura un sous-titre
AX, AY	Integer	Coordonnée du coin supérieur gauche
NbX, NbY	Byte	Nombre de cases en largeur et en hauteur
DimX, DimY	Byte	Dimension (pixels) d'une case en largeur et en hauteur

`function CasePointee`

Objet : retourner le numéro de la case pointée.

Entrée :

Msg	TMessage	Contient entre autres les coordonnées du pointeur de la souris

Sortie :

- numéro de la case où se trouve le pointeur de la souris;
- -1 si le pointeur ne se trouve pas sur une case (il est hors de la boîte, dans une barre de titre ou dans les marges de la boîte).

`procedure CoordCase`

Objet : retourner les coordonnées de la case d'un numéro donné.

Entrée :

NoCase	Integer	Numéro d'une case

Sortie :

Rect	TRect	Coins supérieur gauche et inférieur droit de la case

`procedure SelectionneCase`

Objet : ajouter ou supprimer un cadre autour d'une case de numéro donné.

Entrée :

DC	HDC	Display Context valable pour la fenêtre
NoCase	Integer	Numéro de la case à encadrer
Oui	Boolean	Indique s'il faut dessiner ou effacer le cadre

`function TotalCases`

Objet : retourner le nombre total de cases.

Sortie : nombre de cases de la boîte (numéro de la case inférieure droite + 1).

TBIcônes

Sémantique

TBoîteCases servant à afficher et utiliser la représentation pictographique d'un nombre. L'objet reconnaît 4 ordres de grandeur et toute base comprise entre 2 et 16 (la base choisie est affichée en toutes lettres dans la barre de sous-titre). Selon le comportement assigné à l'objet, l'utilisateur peut ouvrir des icônes et en regrouper.

Icônes

Si B désigne la base numérique, l'ordre de grandeur

- B^0 sera représenté par un bonbon
- B^1 un sachet
- B^2 une caisse
- B^3 une armoire

En outre, pour chaque ordre de grandeur, un code de couleur permet de distinguer différentes catégories d'objets :

Catégorie	*Identificateur*	*Couleur utilisée pour l'objet*
Positif, normal	NPos	bleu sur fond blanc
Positif, report	Report	vert sur fond blanc
Positif, résultat	Resultat	noir sur fond blanc
Positif, erreur	EPos	comme NPos; l'objet est surmonté d'un '?' rouge
Négatif, normal	NNeg	blanc sur fond bleu
Négatif, erreur	ENeg	comme NNeg; l'objet est surmonté d'un '?' rouge

Affichage

Si l'ensemble à afficher comporte des icônes de types différents, elles seront triées selon leur ordre de grandeur. Ainsi seront regroupées d'abord les armoires, puis les caisses, etc. À l'intérieur d'un même ordre de grandeur, la priorité sera : Resultat, NNeg, ENeg, EPos, NPos, Report.

Cet ordre peut être appliqué dans plusieurs directions :

- En colonnes

Un nombre est affiché colonne par colonne, les armoires se trouvant le plus à gauche, les bonbons à droite. Si le nombre d'icônes le permet, le programme passera à la colonne suivante pour chaque nouvel ordre de grandeur. Les piles d'objets reposent sur le fond de la boîte.

Remarque :

Si le nombre à représenter en icônes comporte un zéro, le programme ne laissera en aucun cas une colonne vide pour figurer l'ordre de grandeur absent. Je le justifie par deux arguments.

1. La représentation pictographique ainsi définie est cohérente. Si la numération utilise le 0, c'est parce que les mêmes chiffres servent pour tous les ordres de grandeur. Dans la représentation pictographique, laisser un espace vide n'apporte rien : la position d'un objet n'est pas un critère distinctif. À la limite, on pourrait mélanger toutes les icônes, le nombre représenté ne changerait pas d'un chouïa.
2. Mieux vaut éviter toute réponse automatique chez l'enfant. Si on laissait un espace vide pour un 0, les ensembles de pictogrammes seraient plus proches de l'écriture en chiffres. Le but est bien de faire "ressentir" les nombres, mais cette simplification aurait un inconvénient pédagogique.

   Prenons un exemple. L'ordinateur montre un ensemble de pictogramme et demande à l'enfant d'écrire le nombre qui y correspond. Supposons que le programme ait choisi 1026. Si l'enfant voit l'espace vide entre les armoires et les sachets, il va sans doute répondre juste, mais sans réfléchir. Par contre, si les colonnes se tiennent l'une à l'autre, l'élève qui n'aurait pas encore bien compris le système décimal a des chances de se tromper. Et il faut qu'il se trompe pour pouvoir se corriger.

- En cordes

La seule différence avec l'affichage en colonnes est que chaque pile est "suspendue" au bord supérieur de la boîte au lieu de poser sur le fond.

- Horizontale

Un nombre est affiché ligne par ligne, les armoires se trouvant le plus au-dessus, les bonbons dans le fond. Si le nombre d'icônes le permet, le programme passera à la ligne pour

chaque nouvel ordre de grandeur. Les barres d'objets sont appuyées au bord gauche de la boîte. Le programme ne laisse pas de ligne vide.

Remarque :

Sauf demande explicite, cette orientation est choisie par défaut. En effet, elle supprime encore un automatisme. L'enfant est obligé de réfléchir pour passer du pictogramme (à chaque ordre sa ligne) au nombre (à chaque ordre sa colonne).

Utilisation de la souris

Action logique	*Action physique*
Sélectionner un objet	Positionner le pointeur sur l'objet, cliquer et relâcher le bouton gauche.
Sélectionner un groupe d'objets	Cliquer du bouton gauche (sans relâcher) sur un objet, déplacer le pointeur, relâcher.
Déselectionner un ou plusieurs objets	Procéder comme pour la sélection.
Regrouper des objets	B étant la base numérique, sélectionner B objets de même ordre de grandeur et de même signe.
Détruire des objets	Sélectionner en nombre égal des objets de signes opposés de même ordre de grandeur.
Désélectionner tout	Enfoncer la touche Shift et le bouton gauche de la souris
Ouvrir un objet	Cliquer une fois avec le bouton droit ou deux fois avec le gauche

Remarques :

- Par défaut, l'utilisateur ne peut sélectionner que des objets d'un même ordre de grandeur (si des objets sont sélectionnés et que l'utilisateur clique sur une icône d'un autre ordre de grandeur, seule cette dernière sera encadrée).
- Aucune action n'est possible sur une icône de type Resultat.
- Il est possible de spécifier quels ordres de grandeur peuvent être sélectionnés ou désélectionnés.
- Dans un ordre de grandeur donné, on ne peut sélectionner à la fois des icônes de catégorie EPos et d'une autre catégorie. Idem pour les icônes ENeg.

Emplois

La boîte à icônes est prévue pour 4 contextes.

1. Comportement statique

La boîte se contente d'afficher un nombre, elle ne réagit pas aux clicks de la souris.

2. Appréciation des ordres de grandeur

L'utilisateur peut ouvrir ou regrouper des icônes. L'objet qui résulte de l'ouverture ou du regroupement est de même catégorie (NPos, NNeg, ...) que l'objet d'origine.

Catégorie d'origine	*Catégorie si regoupement*	*Catégorie si ouverture*
Resultat	(refusé)	(refusé)
NNeg	NNeg	NNeg
ENeg	NNeg	(refusé)
EPos	NPos	(refusé)
NPos	NPos	NPos
Report	NPos	NPos

3. Addition

L'utilisateur peut ouvrir ou regrouper des icônes. Le vert est pris comme couleur conventionnelle du report (regroupement). Ainsi, des bonbons bleus cèderont la place à un sachet vert. Un double click sur ce sachet vert rendra les bonbons bleus.

Catégorie d'origine	*Catégorie si regoupement*	*Catégorie si ouverture*
Resultat	(refusé)	(refusé)
NNeg	NNeg	ENeg
ENeg	NNeg	(refusé)
EPos	NPos	(refusé)
NPos	Report	NPos
Report	Report	NPos

4. Soustraction

Ici aussi, le vert est la couleur du report, mais le report correspond cette fois à l'ouverture d'un objet positif et non au regroupement.

L'utilisateur peut annuler des objets positifs et négatifs de même ordre de grandeur. Pour cela, il faut sélectionner dans un seul ordre de grandeur le même nombre d'objets, d'une part dans la catégorie NNeg, d'autre part dans les catégories NPos et Report.

Catégorie d'origine	*Catégorie si regoupement*	*Catégorie si ouverture*
Resultat	(refusé)	(refusé)
NNeg	NNeg	ENeg
ENeg	NNeg	(refusé)
EPos	NPos	(refusé)
NPos	NPos	NPos
Report	NPos	NPos

Champs

Identificateur	*Type*	*Sémantique*
Inter	TPoint	Marge entre le bord d'une case et l'icône qu'elle contient
Card	TCardinal*	Nombre d'icônes de chaque ordre et de chaque catégorie
Base	Byte	Base numérique du nombre représenté
Contenu	PTableauIcones*	Vecteur qui décrit le contenu de chaque case de la boîte
MonoSelect	Boolean	Indique si l'utilisateur ne peut sélectionner qu'un ordre de grandeur à la fois (VRAI par défaut)
NbSelect	TNbSelect*	Nombre d'icônes sélectionnées de chaque ordre et de chaque catégorie
Orientation	VAffichage*	Orientation de l'affichage (cf. supra)
PlumeLimite	HPen	Plume pour tracer la limite de l'objet correspondant au plus grand chiffre de la base choisie
LigneLimite	TRect	Coordonnée de cette ligne
OKLimite	Boolean	Indique si cette ligne doit être tracée
OKSelect	Set of VObjet*	Ensemble des ordres de grandeur dont l'utilisateur peut sélectionner des objets
OKDeselect	Set of VObjet*	Ensemble des ordres de grandeur dont l'utilisateur peut désélectionner des objets
OKOuvrir	Set of VObjet*	Ensemble des ordres de grandeur dont l'utilisateur peut ouvrir des objets
OKFermer	Set of VObjet*	Ensemble des ordres de grandeur dont l'utilisateur peut regrouper des objets
SiOuvrir	Array[VCategorie] of VCategorie*	Catégories obtenues par les différentes ouvertures possibles (cf. supra)
SiFermer	Array[VCategorie] of VCategorie*	Catégories obtenues par les différents regroupements possibles (cf. supra)

Méthodes

procedure AfficheContenu

Objet : afficher la représentation pictographique décrite dans *Contenu*.

Entrée : voir la spécification de l'objet parent

function AffichageOK

Objet : retouner le type de présentation possible, selon l'orientation choisie, la taille de la boîte et le nombre d'icônes à afficher.

procedure AfficheCase

Objet : afficher la case dont le numéro est donné en paramètre.

Entrée :

DC	HDC	Display Context valable pour la fenêtre
NoCase	Integer	Numéro de la case à afficher

function CumulSelect

Objet : Retourner le nombre total d'icônes sélectionnées dans un ordre de grandeur pour un ensemble de catégories.

Entrée :

UnObjet	VObjet*	Ordre de grandeur
Genres	TSetCategories*	Ensemble de catégories

procedure DetailleSelection

Objet : fournir le nombre d'icônes sélectionnées dans chaque catégorie d'un ordre de grandeur donné.

Entrée :

UnObjet	VObjet*	Ordre de grandeur

Sortie :

Detail	TDetailSelect*	Matrice du nombre d'icônes sélectionnées, par catégorie. Detail[Total] est le nombre total d'icônes sélectionnées, toutes catégories confondues (Resultat..Report).

function LargeurSelection

Objet : retourner le nombre d'ordres de grandeurs dans lesquels des icônes sont sélectionnées.

procedure MessageGerant

Objet : envoyer au gérant un message contenant les indications données en paramètes.

Entrée :

ID	Word	Type de message
UnObjet	VObjet*	Ordre de grandeur concerné
UnGenre	VCategorie*	Catégorie de l'icône concernée
Code	Word	Appréciation de l'action tentée par l'utilisateur

function PrendValeur

Objet : afficher la représentation pictographique d'un nombre donné dans une base donnée.

Entrée :

Val	Integer	Nouvelle valeur à convertir en pictogrammes
BaseNum	Byte	Nouvelle base numérique

Sortie :

VRAI si le nombre de cases de la boîte est suffisant pour afficher *Val* en *BaseNum*;

FAUX sinon (la base et le cardinal courants sont conservés).

function PrendCardinal

Objet : afficher un ensemble d'icônes donné (même remarque que pour la métode précédente).

Entrée :

N	TCardinal*	Nombre d'icônes de chaque ordre et catégorie

BaseNum	Byte	Base numérique

Sortie :

VRAI si le nombre de cases de la boîte est suffisant pour afficher toutes les icônes;

FAUX sinon (la base et le cardinal courants sont conservés).

`procedure RegroupeSelection`

Objet : selon les icônes sélectionnées et l'usage assigné à la boîte, procéder à une suppression ou un regroupement éventuels. En signaler le résultat ou le refus au gérant. Cette méthode est appelée automatiquement par WMlButtonUp.

`procedure SelectionneCase`

Objet : sélectionner ou déselectionner une case.

Entrée :

DC	HDC	Display Context valable pour la fenêtre
NoCase	Integer	Numéro de la case visée
Oui	Boolean	VRAI ssi il faut sélectionner la case

`function SelectionUnique`

Objet :

- si les icônes sélectionnées appartiennent toutes au même ordre de grandeur, retouner cet ordre;
- s'il s'agit de plusieurs ordres ou si rien n'est sélectionner, retourner *Rien*.

`procedure Usage`

Objet : assigner à la boîte un nouveau comportement

Entrée :

UnUsage	VUsage*	Cf. supra

`function Valeur`

Objet : retourner la valeur représentée en pictogrammes. Si le contenu comprend des objets positifs et négatifs, la méthode calcule la différence.

procedure VerrouilleObjet

Objet : donner à toutes les icônes d'un ordre de grandeur la catégorie *Resultat*.

Entrée :

UnObjet	VObjet*	Ordre de grandeur

procedure VideCardinal

Objet : détruire et effacer le contenu de la boîte.

procedure VideContenu

Objet : détruire le contenu de la boîte sans l'effacer. L'objet ne répond plus aux clicks de la souris tant qu'un nouveau cardinal n'a pas été reçu.

procedure lButtonDown

procedure lButtonDblClk

procedure lButtonUp

procedure lButtonShift

procedure MouseMove

procedure rButtonDown

Objet : cf. supra ("Utilisation de la souris"). L'objet TBIcones fournit au gérant un message commenté pour :

- toute modification du contenu qui résulte d'un click ou déplacement de la souris;
- toute action logique refusée.

Entrée : cf. objet parent.

TBAddSub

Sémantique

TBoîteCases qui permet d'afficher l'addition ou la soustraction de deux nombres entiers comportant au plus 4 chiffres, dans une base comprise entre 2 et 16.

Cet objet respecte les conventions de couleurs adoptées pour la boîte à icônes : les deux nombres à additionner vs. soustraire sont affichés en bleu, les reports en vert et le résultat en noir. Une erreur commise à un report ou à un chiffre de résultat sera marquée par un point d'interrogation rouge.

Champs

Contenu	Array[0..24] of TMiniChaine*	Chaîne (max. 2 caractères) contenue dans chaque case
CaseSelect	Set of 0..24	Ensemble des numéros des cases sélectionnées
OKSelect	Set of 0..24	Numéros des cases que l'utilisateur peut sélectionner
OKDeselect	Set of 0..24	Numéros des cases que l'utilisateur peut désélectionner
EncreNormal	TColorRef	Couleur d'encre des deux nombres
EncreReport	TColorRef	Couleur d'encre du report
EncreResultat	TColorRef	Couleur d'encre du résultat
EncreErreur	TColorRef	Couleur d'encre d'une erreur (point d'interrogation)
EncreSymbole	TColorRef	Couleur d'encre du symbole et de la barre d'opération
Calcul	TCalcAddSub*	Enregistrement des chiffres corrects du résultat et des reports

Méthodes

`procedure AccepteRang`

Objet : autoriser ou interdire la sélection des cases appartenant à un ordre de grandeur donné.

Entrée :

Rang VObjet* Ordre de grandeur

procedure AfficheContenu

Objet : dessiner le calcul écrit (les deux nombres, les chiffres de report et de résultat enregistrés, le symbole et la barre de l'opération).

Entrée : cf. objet parent.

procedure DetermineReports

Objet : effectuer le calcul écrit correct sans rien afficher, stocker le résultat et les reports.

procedure PrendValeurs

Objet : fixer les deux nombres et l'opération à effectuer. Effecteur le calcul écrit correct. Afficher le problème en laissant vide les zones de report et de résultat.

Entrée :

Val1, Val2	Integer	Les deux nombres du problème
B	Byte	Base numérique des deux nombres
Op	Char	Symbole de l'opération à effectuer ('+' ou '-')

Préconditions :

- $(0 <= Val1, Val2 <= 9999_B)$ et $(2 <= B <= 16)$;
- Si Op = '+', le résultat Val1 + Val2 doit comprendre au plus 4 chiffres dans la base B;
- Si Op = '-', le résultat Val1 - Val2 doit être positif et comprendre au plus 4 chiffres dans la base B.

procedure PrendReport

Objet : afficher une valeur de report à un rang donné. La méthode compare le nombre donné à celui attendu. Elle affiche le paramètre s'il est correct, ou un point d'interrogation sinon.

Entrée :

Rang	VObjet*	Rang auquel le report doit être fait
Rep	Byte	Valeur du report (1 dans une addition, 10 dans une soustraction).

procedure PrendResultat

Objet : afficher un chiffre de résultat à un rang donné. La méthode compare le chiffre de résultat donné à celui attendu. Elle affiche le chiffre si le paramètre est correct, ou un point d'interrogation sinon.

Entrée :

Rang	VObjet*	Rang auquel le chiffre du résultat doit s'afficher
Rep	Byte	Chiffre du résultat

procedure SelectionneCase

Objet : ajouter ou supprimer la mise en évidence d'une case.

Entrée : cf. objet parent

procedure SelectionneRang

Objet : ajouter ou supprimer la mise en évidence des cases d'un rang donné.

Entrée :

Rang	VObjet*	Ordre de grandeur
Oui	Boolean	VRAI ss'il faut mettre en évidence le rang

procedure VideContenu

Objet : détruire et effacer le contenu de la boîte.

procedure lButtonDown

Objet :

- identifier la case pointée.

S'il s'agit bien d'une case :

- voir si elle est sélectionnée ou non et si elle peut être (dé)sélectionnée;
- si nécessaire, mettre en oeuvre la (dé)sélection;
- avertir le gérant du résultat.

Entrée : cf. objet parent

`procedure lButtonUp`

Objet :

- identifier la case pointée (ligne et ordre de grandeur);
- avertir le gérant que le bouton gauche a été relâché sur cette case.

Cette méthode est utilisée pour implémenter le drag-and-drop. Le lecteur en trouvera les détails plus loin, dans la description de l'Exercice 2.

Entrée : cf. objet parent

TBoîteCapture

Sémantique

TBoîteCases permettant de réaliser du drag-and-drop. Cliquer sur une case de la boîte n'opère plus un marquage statique mais un prélèvement. Un cadre en pointillé suit le pointeur de la souris jusqu'au moment où l'utilisateur relâche le bouton.

Champs

Identificateur	*Type*	*Sémantique*
OKCapture	TSetByte*	Numéros des cases dont le contenu peut être prélevé
Curseur	TRect	Coordonnées du cadre qui suit le pointeur de la souris lors du prélèvement
Contour	HBrush	Brosse utilisée pour tracer le pointeur du prélèvement

Méthodes

`procedure CadreXOR`

Objet : déplacer le pointeur du prélèvement et enregistrer sa nouvelle position.

Entrée :

Rect1	TRect	Coordonnées de la position actuelle du pointeur
Rect2	TRect	Coordonnées de la nouvelle position du pointeur

Note : si Rect2 vaut ((0;0);(0;0)), le pointeur n'est pas redessiné.

`procedure MessageGerant`

Objet : envoyer un message au gérant, selon les indications données en paramètres.

Entrée :

ID	Word	Action logique effectuée
NoCase	Integer	Case prélevée
Code	Word	Indique le succès ou l'éche de l'action

procedure lButtonDown

Objet : analyser la validité d'un prélèvement et le mettre en oeuvre si nécessaire. Avertir le gérant du succès ou de l'échec.

Entrée : cf. objet parent

procedure MouseMove

Objet : si un prélèvement est en cours, déplacer le pointeur.

Entrée : cf. objet parent.

procedure lButtonUp

Objet : si un prélèvement est en cours, effacer le pointeur et signaler le dépôt au gérant.

Entrée : cf. objet parent.

TBSimpleAdd

Sémantique

TBoîteCapture destinée à contenir au plus 3 valeurs et leur somme. Les chiffres du résultat peuvent être prélevés (drag-and-drop). L'objet est censé recevoir 3 nombres qui correspondent à une colonne d'addition ou de soustraction écrite : un report (0, 1 ou 10), un chiffre du premier nombre et un du second (0 à 9).

Champs

Contenu	Array[0..7] of TMiniChain*e	Contenu de chaque case
OKValeurs	Set of 1..3	Numéros des chiffres déjà positionnés
Valeur	Array[1..3] of Byte	Valeur des trois chiffres
Somme	Byte	Somme des chiffres positionnés
SommeDonnee	Boolean	Indique si la somme des chiffres est déjà calculée
Operation	Char	Vaut '-' si l'objet affiche une valeur négative, '+' sinon
QuiCalcule	VChoix*	Indique qui doit fournir la somme des chiffres (l'ordinateur ou l'élève)

Méthodes

`procedure AfficheCase`

Objet : afficher le contenu d'une case donnée.

Entrée : cf. objet parent.

`procedure AfficheContenu`

Objet : afficher le contenu de toutes les cases. Les chiffres déjà enregistrés qui composent la somme se placent à gauche, séparés par le signe '+'. Le résultat, s'il a été reçu, s'inscrit à droite, séparé de la somme par le signe '='.

Entrée : cf. objet parent.

`procedure CalculeSomme`

Objet : selon l'option *QuiCalcule*, l'ordinateur inscrit lui-même la somme des chiffres reçus et autorise le drag-and-drop, ou bien il demande le résultat à l'utilisateur (en vérifiant

qu'il est correct). L'objet permet ensuite la sélection du (des) chiffre(s) du résultat pour le drag-and-drop.

procedure MessageGerant

Objet : envoyer un message au gérant, selon les indications données en paramètres. Le message spécifiera s'il s'agit du début (drag) ou de la fin (drop) d'un prélèvement, si l'opération est acceptée ou refusée dans le cas du début.

Entrée :

ID	Word	Type de message
NoCase	Integer	Numéro de la case prélevée
Code	Word	Qualification de l'action (succès ou échec)

procedure AjouteValeur

Objet : placer un chiffre à une position donnée. La somme est effacée.

Entrée :

Cat	VCategorie*	Type de valeur à afficher (*Report* pour la valeur 0, 1 ou 10 du report; *NPos* pour les chiffres des deux nombres d'une addition; *NNeg* pour les chiffres du second nombre d'une soustraction)
UneValeur	Byte	Valeur à afficher

procedure VideContenu

Objet : détruire et effacer le contenu de la boîte.

Fonction d'appoint

FUNCTION DialogueSommeChiffres

Objet : exécuter un dialogue qui demande la somme des chiffres affichés dans la boîte. Si la réponse fournie n'est pas correcte, le dialogue donne un message et redemande la réponse. Au bout de trois essais infructueux, l'objet donne la réponse correcte.

Entrée :

Fenetre	PWindow	Fenêtre principale du programme
Digits	T3Chiffres	3 valeurs affichées dans l'objet TBSimpleAdd
Op	Char	Opération ('+' ou '-')

Sortie :

NbEssais	Byte	Nombres d'essais avant que l'utilisateur donne la bonne réponse

TBouton

Sémantique

Cet objet n'a pour but que de rendre plus transparente l'utilisation des boutons au sein d'une fenêtre, en évitant le recours à des messages et fonctions Windows aux noms irréductibles.

Méthodes

`function DonneTexte`

Objet : retourner le texte inscrit dans le bouton.

`procedure DevientDefaut`

Objet : donner l'aspect de bouton par défaut (entouré d'une épaisse ligne noire).

`procedure DevientNormal`

Objet : rendre au bouton l'état normal (non entouré d'une épaisse ligne noire).

`function EstActif`

Objet : retourner VRAI ssi le bouton réagit à un click de la souris (dans un bouton inactif, le texte est grisé au lieu de noir).

`procedure PrendTexte`

Objet : afficher un texte sur le bouton.

`procedure RecoitFocus`

Objet : accorder ou supprimer le focus (le bouton qui dispose du focus peut être enfoncé par la touche <Espace>), selon le paramètre booléen donné.

`procedure RendActif`

Objet : rendre actif ou inactif le bouton, selon le paramètre booléen donné.

TFenêtreVide

Sémantique

Descendant du type TWindow, destiné à contenir des boîtes et des boutons. TFenêtreVide redirige vers les boîtes concernées les événements générés lors de toute manipulation de la souris.

Champs

Gerant	PGerantVide	Adresse du gérant de l'exercice
FocusGauche	Byte	Numéro de la boîte où le bouton gauche de la souris a été enfoncé
FocusDroit	Byte	Numéro de la boîte où le bouton droit de la souris a été enfoncé
Position	LongInt	Positon du pointeur de la souris
NBBoites	Byte	Nombre de boîtes présentes dans la fenêtre
Boite	TTableauBoites*	Table des adresses des boîtes
NBBoutons	Byte	Nombre de boutons présents dans la fenêtre
Bouton	TTableauBoutons*	Table des adresses des boutons
Parametres	TParametresPrg*	Options générales du didacticiel

Méthodes

`function AjouteBoite`

Objet : ajouter une boîte dans la fenêtre.

Entrée :

Adresse PBoite Adresse de la boîte à ajouter

Sortie : VRAI ssi la boîte a pu être ajoutée

`function AjouteBouton`

Objet : ajouter un bouton dans la fenêtre.

Entrée :

Adresse	PBouton	Adresse du bouton à ajouter

Sortie : VRAI ssi le bouton a pu être ajouté

procedure Message

Objet : afficher une fenêtre de message.

Entrée :

Texte, Titre	PChar	Message à donner à l'utilisateur et titre.

function CanClose

Objet : demander confirmation à l'utilisateur pour quitter l'application.

Sortie : VRAI ssi l'utilisateur confirme.

procedure ChoisitExercice

Objet : afficher tous les boutons créés pour l'exercice.

procedure DebutExercice

Objet : commander au gérant le début de l'exercice avec de nouvelles valeurs.

procedure GetWindowClass

Objet : assigner une classe à la fenêtre principale de l'application.

Sortie :

AWndClass	TWndClass	Descripteur de la classe

procedure OptionsGenerales

Objet : exécuter un dialogue qui présente à l'utilisateur des choix d'options.

procedure Paint

Objet : redessiner tout ou partie de la fenêtre.

Entrée :

DC	HDC	Display Context fourni par Windows pour redessiner le contenu de la fenêtre

Info	TPaintStruct	Structure indiquant entre autres les coordonnées de la région à redessiner

procedure Quitter

Objet : demander la fermeture de la fenêtre principale de l'application.

procedure SetUpWindow

Objet : donner à la fenêtre principale sa taille maximale, la dessiner si ce n'est déjà fait, permettre au gérant de modifier le menu.

procedure ClickBoutonXX

Objet : répondre à un click sur le bouton numéro XX. Il y a une procédure semblable par bouton pouvant être défini : en l'occurrence, XX est à remplacer par les valeurs 01 à 05.

Entrée :

Msg	TMessage	Paramètres du message wm_lButtonDown

procedure WMCancelMode

Objet : méthode appelée lorsque la fenêtre perd le focus (par exemple lorsque s'affiche une fenêtre de message ou une boîte de dialogue). Il s'agit de terminer les actions en cours qui utilisent la souris.

Entrée :

Msg	TMessage	Contient les coordonnées du pointeur de la souris

Réponse aux événements de type wm_Mouse

Les méthodes qui suivent sont chacune dédicacées à un événement souris particulier. Elles sont appelées automatiquement par Windows et redirigent si nécessaire le message reçu vers une ou plusieurs boîtes concernées.

- L'utilisateur enfonce un bouton.

La méthode appelée vérifie si le pointeur se trouve dans une boîte. Si oui, il lui signale l'événement et, selon le bouton enfoncé, enregistre dans FocusGauche ou FocusDroit le numéro de la boîte visée.

- L'utilisateur relâche le bouton droit.

La méthode signale l'événement à la boîte de numéro FocusDroit. FocusDroit est remis à 0.

- L'utilisateur relâche le bouton gauche.

La méthode appelée vérifie si le pointeur se trouve dans une boîte. Si oui, il signale l'événement, d'abord à cette boîte, puis à celle de numéro FocusGauche (dans laquelle le bouton avait été enfoncé). S'il s'agit de la même boîte, WMlButtonUp n'envoie naturellement q'un message. L'envoi d'un double message se justifie par la réalisation du drag-and-drop, dont le lecteur trouvera les détails infra, dans la spécification de l'Exercice 2. FocusGauche est remis à 0.

- L'utilisateur déplace le pointeur de la souris.

Si un bouton de la souris est enfoncé (ce que la méthode détermine en examinant les valeurs de FocusGauche et FocusDroit), WMMouseMove signale le déplacement à la boîte dans laquelle le bouton avait été enfoncé.

procedure WMlButtonDown

Objet : click du bouton gauche dans la zone cliente de la fenêtre.

procedure WMlButtonShift

Objet : click du bouton gauche dans la zone cliente de la fenêtre alors que la touche Shift est enfoncée (méthode appélee par WMlButtonDown).

procedure WMlButtonUp

Objet : relâchement du bouton gauche dans la zone cliente de la fenêtre.

procedure WMNClButtonUp

Objet : relâchement du bouton gauche dans la zone non cliente de la fenêtre.

procedure WMlButtonDblClk

Objet : double click du bouton gauche dans la zone cliente de la fenêtre. Un double click complet génère 4 messages : wm_lButtonDown, wm_lButtonUp, wm_lButton-DblClk, wm_lButtonUp.

procedure WMMouseMove

Objet : déplacement du pointeur de la souris.

procedure WMrButtonDown

Objet : click du bouton droit dans la zone cliente de la fenêtre.

procedure WMrButtonUp

Objet : relâchement du bouton droit dans la zone cliente de la fenêtre.

procedure WMNCrButtonUp

Objet : relâchement du bouton droit dans la zone non cliente de la fenêtre.

Fonction d'appoint

FUNCTION DialogueOptionsPrg

Objet : exécuter un dialogue avec les paramètres généraux du programme (TDlgOptionsPrg).

Entrée :

Fenetre	PWindow	Fenêtre principale du programme

Entrée/Sortie :

Param	TParametresPrg	Paramètres courants du programme

Sortie :

VRAI ssi l'utilisateur a terminé le dialogue en cliquant sur le bouton OK.

Objets sans élément visuel

TCalcAddSub

Sémantique

Objet destiné à calculer le résultat et les reports d'une addition/soustraction de deux nombres entiers, comportant au plus 4 chiffres, dans une base comprise entre 2 et 16.

Champs

Operation	Char	Symbole de l'opération à effectuer ('+' ou '-')
Base	Byte	Base numérique des deux nombres
Nombre	Array[1..2] of TReprNombre*	Les deux nombres à additionner/soustraire
ResultatOK	TReprNombre*	Vecteur des chiffres du résultat
ResultatBrutOK	TReprNombre*	Pour chaque rang, somme des chiffres des deux nombres avant le report éventuel au rang suivant
ResultatVu	TReprNombre*	Vecteur des chiffres affichés au résultat
ReportVu	TReprNombre*	Vecteur des chiffres affichés comme reports
ReportOKDe	Array[Bonbon.. Armoire] of Boolean	Indique, pour chaque rang, s'il est origine d'un report
ReportOKVers	Array[Bonbon.. Armoire] of Boolean	Indique, pour chaque rang, s'il est destination d'un report
RangMax	VObjet*	Rang le plus élevé du résultat

Méthodes

`constructor Init`

Objet : initialiser les champs de l'objet. Operation et Base reçoivent une valeur par défaut, les autres champs une valeur nulle.

`procedure EffectueCalcul`

Objet : calculer le résultat et les reports.

`procedure PrendValeurs`

Objet : initialiser les deux nombres et le symbole de l'opération, puis effectuer les calculs.

Entrée :

Val1, Val2	Integer	Les deux nombres à additionner/soustraire
B	Byte	Base numériques des nombres
Op	Char	Symbole de l'opération ('+' ou '-')

Préconditions :

- (0 <= Val1, Val2 <= 9999_B) et (2 <= B <= 16);
- Si Op = '+', le résultat Val1 + Val2 doit comprendre au plus 4 chiffres dans la base B;
- Si Op = '-', le résultat Val1 - Val2 doit être positif et comprendre au plus 4 chiffres dans la base B.

TGérantVide

Sémantique

Objet générique servant de structure à tous les exercices. La fenêtre principale de l'application (TFenêtreVide) possède à tout moment un gérant qui organise le déroulement d'un exercice.

Champs

Fenetre	PFenetreVide	Adresse de la fenêtre du didacticiel
Stop	Boolean	VRAI ssi l'exercice est ou doit être arrêté

Méthodes

`constructor Init`

Objet : initialiser les champs de l'objet, appler la méthode GarnitMainWindow, commander à la fenêtre de se redessiner. *Normalement, cette méthode ne doit pas être surchargée. La définition de l'écran incombe seulement à GarnitMainWindow.*

`destructor Done`

Objet : détruire les variables dynamiques utilisées; commander à la fenêtre de détruire son contenu.

`procedure Avertissement`

Objet : afficher un texte dans une boîte de message.

Entrée :

ID	Integer	Identifiant de la chaîne de caractères (contenue dans les ressouces du programme) à afficher dans la boîte

`procedure ChangeParametres`

Objet : prendre connaissance des options générales du programme, détenues par la fenêtre.

procedure ClickBouton

Objet : répondre à un click sur le bouton dont le numéro est donné en paramètre.

Entrée :

No	Byte	Numéro de bouton

procedure DoneExercice

Objet : clôturer l'exercice

procedure GarnitMainWindow

Objet : placer dans la fenêtre les boîtes et boutons nécessaires à l'exercice. *Aucune demande d'affichage n'est nécessaire dans cette méthode.*

procedure InitExercice

Objet : commencer un exercice avec de nouvelles valeurs.

Entrée :

Param	Pointer	Réservé à des spécialisations ultérieures.

procedure RecoitMessage

Objet : traiter un message généré par une boîte.

Entrée :

Msg	TMsgDidact*	Description du message

procedure VideMainWindow

Objet : détruire les boîtes et boutons contenus dans la fenêtre.

Préconditions :

- le vecteur Boite[1..NbBoites] contient des pointeurs PBoite valides;
- le vecteur Bouton[1..NbBoutons] contient des pointeurs PBouton valides.

TDidactVide

Sémantique

Application dont la fenêtre principale est une TFenêtreVide.

Champ

Icone	HIcon	Icône de l'application

Méthodes

`constructor Init`

Objet : initialiser l'application.

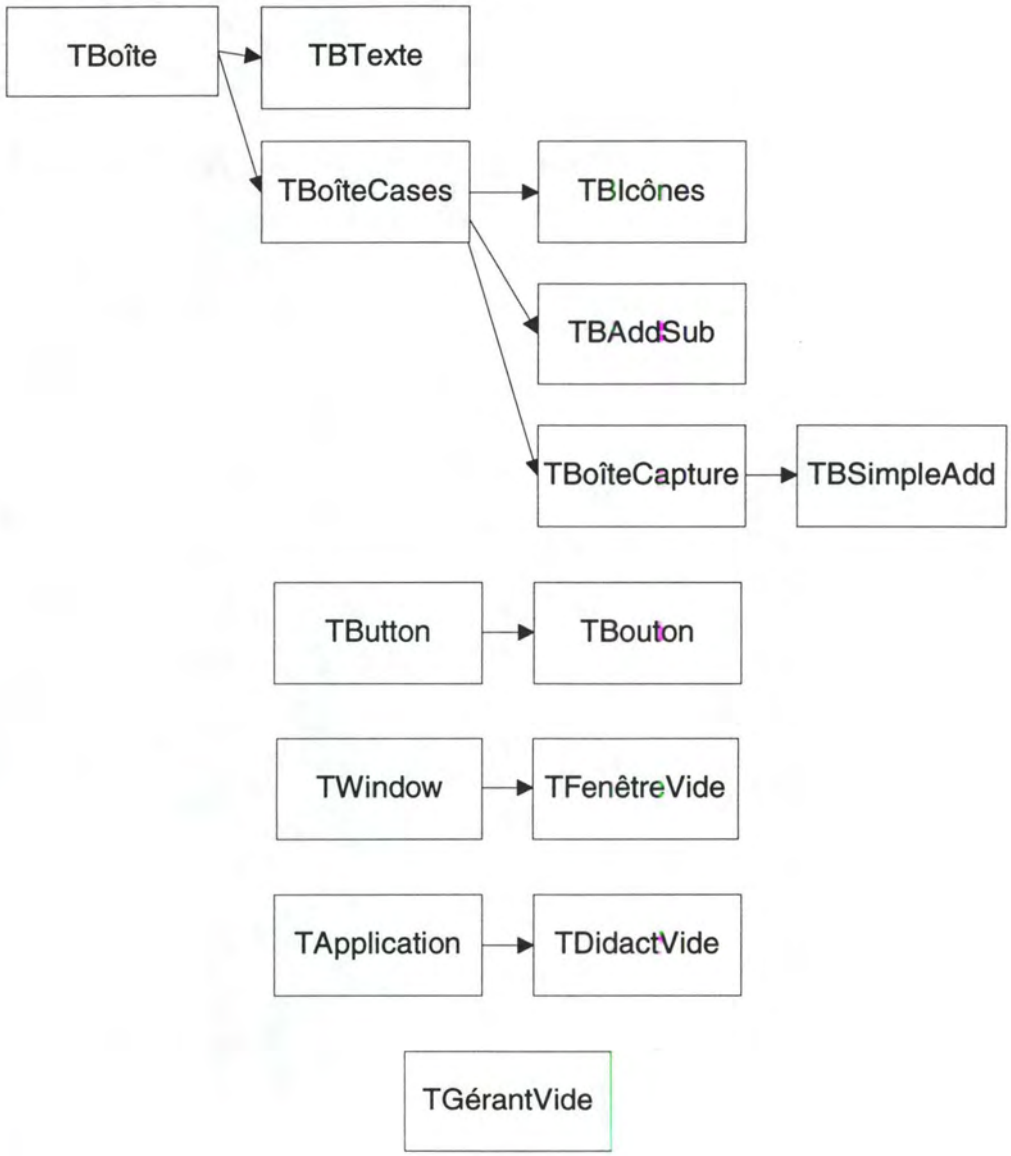
Entrée :

AName	PChar	Nom de l'application
AnIcon	PChar	Nom de l'icône de l'application (contenue dans les ressources)

`procedure InitMainWindow`

Objet : créer la fenêtre principale de l'application.

Hiérarchie des objets



Procédures et fonctions d'appoint

PROCEDURE ChargeIcones

Objet : placer dans une matrice (*StockIcone*, variable globale) les identifiants des icônes qui servent à représenter un nombre. StockIcone a deux dimensions : l'une est l'ordre de grandeur, l'autre la catégorie.

PROCEDURE ConvertitNombre

Objet : déterminer les différents chiffres qui composent l'écriture d'un nombre dans une base; stocker les derniers chiffres du nombre dans un tableau.

Entrée :

N	Integer	Valeur à convertir
B	Byte	Base numérique du nombre de destination
T		Adresse d'un tableau destiné à recevoir les C derniers chiffres
C	Byte	Nombre de chiffres à copier

PROCEDURE CreeBrosseSolide

Objet : créer une brosse de couleur unie donnée.

Entrée :

Couleur	TColorRef	Couleur de la brosse à créer
Adresse	HBrush	Handle d'une brosse à détruire (ignoré si 0)

Sortie :

Adresse	HBrush	Handle de la brosse créée

Précondition :

Si Adresse <> 0, elle désigne une brosse valide.

PROCEDURE CreePlumeSolide

Objet : créer une plume servant à tracer un trait continu de couleur donnée.

Entrée :

Couleur	TColorRef	Couleur du trait
Adresse	HBrush	Handle d'une plume à détruire (ignoré si 0)

Sortie :

Adresse	HBrush	Handle de la plume créée

FUNCTION Egalite

Objet : déterminer si deux zones de la mémoire contiennent des données identiques.

Entrée :

A, B		Adresse de départ de deux zones en mémoire
Size	Word	Taille des zones à comparer

Précondition :

Si Adresse <> 0, elle désigne une plume valide.

FUNCTION EnToutesLettres

Objet : fournir une chaîne qui est l'écriture en toutes lettres (en français) d'un nombre entier.

Entrée :

Valeur	LongInt	Nombre à écrire

PROCEDURE MetAZero

Objet : remplir une zone de la mémoire centrale avec l'octet nul.

Entrée :

A		Adresse de départ d'une zone en mémoire centrale
Taille	Word	Nombre d'octets à mettre à 0

PROCEDURE Remplit

Objet : remplir une zone de la mémoire centrale avec la répétition d'une suite d'octets.

Entrée :

A		Adresse de départ d'une zone en mémoire centrale
Taille	Word	Nombre d'octets à changer
Sequence	String	Pattern à utiliser pour le remplissage

PROCEDURE TailleBouton

Objet : déterminer la taille minimum d'un bouton pour qu'il puisse contenir un texte donné.

Entrée :

Fenetre	PWindow	Adresse de la fenêtre qui contiendra le bouton
Texte	PChar	Texte qui sera inscrit dans le bouton
dX, dY	Integer	Largeur et hauteur du bouton (en pixels)

Utilisation des objets dans un programme

Exercice 1

Description générale

Deux nombres sont représentés sous la forme de pictogrammes canoniques. L'élève réunit les deux ensembles d'objets. Il effectue les regroupements nécessaires pour que le résultat puisse être écrit en chiffres. L'ordinateur complète le calcul écrit au fur et à mesure.

Concepts

L'addition est une réunion d'ensembles.

Le cardinal d'un ensemble s'écrit en chiffres. Il doit en être de même pour le cardinal d'une réunion d'ensembles, d'où certains reports.

En opérant dans l'ordre, d'abord par les unités, on effectue le moins de manipulations possible.

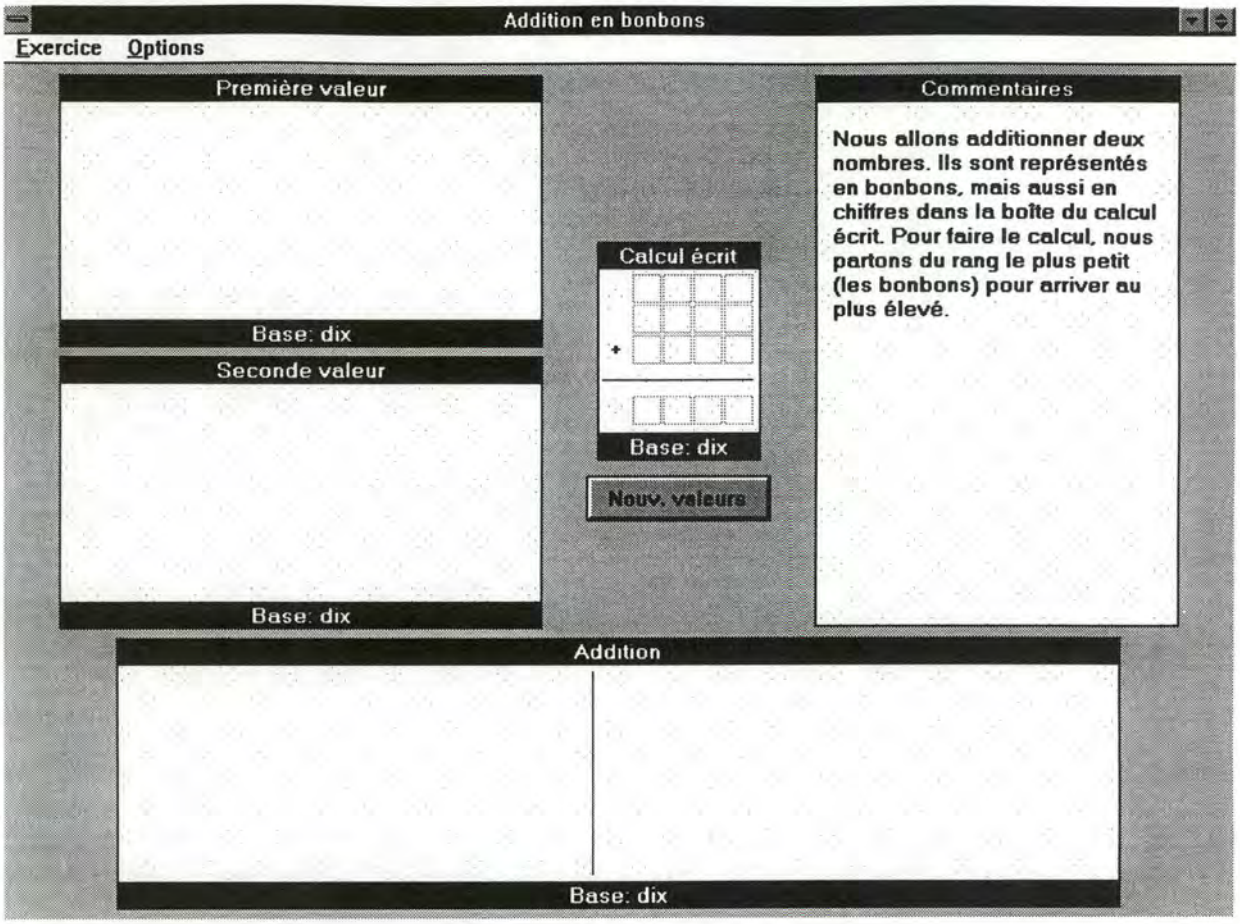
Description de l'écran

Boîtes

1. PREMIÈRE VALEUR : boîte à icônes qui représente le premier nombre.
2. SECONDE VALEUR : boîte à icônes qui représente le second nombre.
3. ADDITION : boîte à icônes qui contient la réunion des ensembles 1 et 2, résultat chiffrable après manipulation.
4. CALCUL ÉCRIT : l'ordinateur y complète l'addition chiffrée au fur et à mesure.
5. INSTRUCTIONS : boîte de texte où le programme explique ce que l'utilisateur doit faire.

Bouton

1. NOUV. VALEURS : bouton qui permet à l'élève de commencer une nouvelle addition.



Déroulement de l'exercice

INITIALISATION

L'utilisateur clique sur le bouton "Nouv. valeurs" pour obtenir deux nombres. Ils sont affichés en bonbons dans leurs cadres respectifs et en chiffres dans le calcul écrit.

BOUCLE

L'addition s'opère rang par rang, en commençant par les bonbons. Pour chaque ordre de grandeur, l'opération se déroulement comme suit :

1. L'utilisateur sélectionne dans les deux ensembles à additionner tous les objets du rang courant R.
2. Le programme ajoute à l'ensemble d'addition une copie de ces objets.
3. Si, dans l'ensemble d'addition, le nombre d'objets R dépasse 9, l'élève effectue un regroupement. L'ordinateur signale et affiche le report au rang suivant.
4. L'élève sélectionne les objets R qui restent. Le chiffre correspondant s'affiche à la bonne place dans le calcul écrit.

TERMINAISON

L'addition est terminée lorsque :

- dans les deux ensembles initiaux, tous les objets de tous les rangs ont été sélectionnés;
- dans la boîte d'addition, pour chaque rang, le regroupement nécessaire a été fait et le reste des objets sélectionné.

Choix des nombres

Selon l'option choisie, les valeurs sont déterminées aléatoirement par l'ordinateur ou demandées au pupitreur. L'utilisateur peut choisir au préalable le nombre maximum de chiffres dans le menu *Options*. La procédure de choix vérifie que les nombres répondent à ce critère de taille et que leur somme ne dépasse pas 9999.

Gestion des erreurs

Ignorant la nature de l'exercice, nous pouvons classer les erreurs en distinguant deux cas de figure.

1) Un objet sait que telle ou telle action est interdite. Par exemple, si le résultat est déjà calculé pour les unités, il ne faut plus sélectionner les bonbons dans les deux ensembles initiaux. L'objet peut transmettre au gérant un message d'erreur.

2) Un objet accepte une action que seul le gérant sait incorrecte. La justification pédagogique est triple.

- L'élève en difficultés ne résoudra pas tous ses problèmes si on ne lui laisse faire que des choses correctes. Il risquerait d'apprendre par coeur un raisonnement qu'il ne comprend pas tout à fait.
- Par contre, s'il peut effectuer certaines opérations fautives, il en voit le résultat et comprend sans doute déjà pourquoi ça cloche. En outre, l'exerciseur lui fournit des explications adaptées. S'il a compris son erreur, il ne sera plus tenté de la commettre.
- Enfin, il peut corriger son action tout de suite et lie mieux à l'erreur le comportement correct et sa justification théorique.

Instructions données pendant l'exercice

L'ordinateur donne des explications à l'élève aux différentes phases de l'exercice. A chaque situation correspond une constante. Son identificateur commence par cg_ (pour *Commentaire du Gérant)*, et sa valeur sert à identifier la chaîne de caractères à afficher, contenue dans les ressources du programme.

Les constantes *cg_Début* et *cg_Fin* désignent directement la ressource à utiliser. Pour les situations qui se rapportent au traitement d'un rang, un texte de commentaire a été prévu pour chaque ordre de grandeur. Le numéro de la chaîne à employer s'obtient par la somme de deux nombres :

- la constante propre à la situation;
- le numéro de l'ordre de grandeur concerné (de 1 pour les unités à 4 pour les milliers).

La liste des littéraux est reprise ci-dessous. Pour chaque constante, je décris la situation correspondante et donne un exemple de commentaire, pour un ordre de grandeur quelconque.

DÉBUT ET FIN DE L'EXERCICE

`cg_Début`

Situation	*Texte unique*
Initialisation de l'exercice, avant même que l'élève ou l'ordinateur ne choisisse les deux premières valeurs.	"Nous allons additionner deux nombres. Ils sont représentés en bonbons, mais aussi en chiffres dans la boîte du calcul écrit. Pour faire le calcul, nous partons du rang le plus petit (les bonbons) pour arriver au plus élevé."

`cg_Fin`

L'addition des deux valeurs est terminée.	"Voilà, l'addition est terminée. Pour choisir deux nombres et les additionner, clique sur le bouton <Valeurs>. Eventuellement, consulte le menu Options | Générales | Qui choisit les valeurs ? ..."

INITIALISATION D'UN RANG

`cg_YaPasDe`

Situation	*Exemple de texte*
Ni les deux ensembles initiaux, ni la boîte d'addition ne contiennent un objet de l'ordre de grandeur courant.	"[VALEURS EN BONBONS] Puisqu'il n'y a pas de caisse, nous inscrivons 0 comme résultat au 3e rang."

`cg_ReportSeul`

Les deux ensembles initiaux ne contiennent aucun objet de l'ordre de grandeur courant, au contraire du cadre d'addition.	"[VALEURS EN BONBONS] Les deux nombres ne comprennent aucun sachet. [ADDITION]Par contre, le report que tu viens d'effectuer doit figurer comme résultat. Sélectionne l'unique sachet pour passer au rang suivant."

`cg_CopieTout`

Il y a au moins un objet de cet ordre de grandeur.	"[VALEURS EN BONBONS] Dans les deux nombres, sélectionne tous les objets de type 'bonbon'. Ils seront ensuite copiés pour effectuer l'addition."

OPÉRATIONS SUR LE RANG COURANT

`cg_TraiteRang`

Le rang courant vient de changer. L'élève doit voir lui-même si un report est nécessaire.	"[ADDITION] Si nécessaire, regroupe des bonbons en un sachet. Puis sélectionne les bonbons restants comme résultat du rang."

cg_FaisReport

Un regroupement est nécessaire.	"[ADDITION] En base 10, nous ne pouvons écrire que des chiffres de 0 à 9. Or, le nombre de bonbons est supérieur à neuf. Regroupe dix bonbons en un sachet."

cg_ReportFait

Un regroupement a été fait.	"[CALCUL ECRIT] Bien ! Tu as regroupé les bonbons, un sachet apparaît. Dans le calcul, le remplacement effectué se marque par un report de 1 au rang des dizaines (sachets)."

cg_SélectReste

Le regroupement fait, il reste des objets du rang courant.	"[ADDITION] \r\nSélectionne tous les sachets : le chiffre qui y correspond sera le résultat de l'addition au rang 2."

cg_YaPlusDe

Une fois le regroupement fait, il ne reste plus d'objet du rang courant.	"[ADDITION] Comme il n'y a plus de caisse, le chiffre de la somme au rang 3 est 0."

Erreurs possibles

Les avertissements en cas d'erreurs sont tous stockés dans un fichier de ressouces (StringTable). L'utilisation des littéraux suit le même principe que pour les commentaires. Toute erreur peut être commise pour n'importe quel ordre de grandeur. Pour chaque situation, quatre chaînes ont donc été stockées. L'identifiant correct s'obtient par la somme de deux valeurs :

- la constante de forme ag_xxxx (pour *Avertissement du Gérant*);
- le numéro de l'ordre de grandeur concerné (de 1 pour les bonbons à 4 pour les armoires).

EXERCICE TERMINÉ

`cg_Valeurs`

Cause	*Réponse*
L'élève clique dans une des boîtes à icônes alors que l'addition est terminée.	"Pour choisir deux nombres et les additionner, clique sur le bouton <Nouv. valeurs>. Eventuellement, consulte le menu Options | Générales | Qui choisit les valeurs ? ..."

PREMIÈRE et SECONDE VALEURS

`ag_PourquoiDésélectNombre`

Cause	*Exemple de réponse pour un rang*
L'élève veut désélectionner des objets du rang courant.	"Tu avais bien commencé. Les bonbons ne seront copiés que quand tu les auras sélectionnés TOUS DANS LES DEUX NOMBRES."

`ag_RefusOuvertureNombre`
`ag_OuvertureInutileNombre`

L'élève tente d'ouvrir un bonbon	"Nous n'utilisons que des nombres entiers. Donc, comme le bonbon est l'objet le plus petit, tu ne peux pas l'ouvrir."

L'élève tente d'ouvrir un sachet, une caisse ou une armoire.	"Si tu ouvrais le sachet, tu aurais 10 bonbons de plus. Mais comme le plus grand chiffre est 9, tu ne peux pas avoir plus de 9 bonbons, sinon tu ne peux pas écrire le nombre."

ag_RangDéjàFait

Alors que le rang courant est R, l'élève tente de sélectionner un objet de rang R' < R.	"Tu as déjà additionné les sachets. Le résulat aux dizaines ne changera pas."

ag_RangPasEncore

Alors que le rang courant est R, l'élève tente de sélectionner un objet de rang R' > R.	"Tu pourrais calculer la somme des sachets avant celle des bonbons. Mais, après, si tu avais 10 bonbons à regrouper en un sachet, tu devrais changer le résulat des dizaines."

ag_NombresDéjàCopiés

Alors que tous les objets du rang courant sont sélectionnés et copiés pour l'addition, l'élève clique à nouveau sur l'un d'eux.	"Tu as déjà copié toutes les caisses. Maintenant, tu dois travailler dans le rectangle d'addition. Pour faire un report, sélectionne EXACTEMENT 10 caisses. Si le report est déjà fait, ou bien s'il n'y a pas plus de 9 caisses, sélectionne-les toutes."

ADDITION

ag_PasEncoreToutAbaissé

L'élève veut traiter le rang courant alors qu'il n'a pas encore sélectionné tous les objets de même ordre dans les deux ensembles initiaux.	"Tu as oublié des sachets dans les deux nombres à additionner. Tant que tu ne les auras pas sélectionnés tous, ils ne s'afficheront pas dans le rectangle d'addition. "

ag_OuvertureInutileSomme

L'élève tente d'ouvrir un bonbon	"Nous n'utilisons que des nombres entiers. Donc, comme le bonbon est l'objet le plus petit, tu ne peux pas l'ouvrir."
L'élève tente d'ouvrir un sachet, une caisse ou une armoire.	"Utiliser une armoire, c'est plus pratique que 10 caisses. Et comme les chiffres vont de 0 à 9, tu ne peux pas écrire le nombre si tu as plus de 9 caisses. Regroupe ces caisses pour retrouver une armoire."

ag_RefusOuvertureSomme

L'élève tente d'ouvrir un bonbon	"Nous n'utilisons que des nombres entiers. Donc, comme le bonbon est l'objet le plus petit, tu ne peux pas l'ouvrir."
L'élève tente d'ouvrir un sachet, une caisse ou une armoire.	"Cette armoire vaut dix caisses. Si tu l'ouvrais, tu aurais 10 caisses de plus. Mais comme les chiffres vont de 0 à 9, tu ne pourrais pas écrire le nombre."

ag_PourquoiDésélectSomme

L'élève veut désélectionner des objets du rang courant.	"S'il y a plus de 9 bonbons, tu en sélectionnes EXACTEMENT 10 pour faire un report. Si le report est déjà fait, ou bien s'il n'y a pas plus de 9 bonbons, sélectionne-les tous."

ag_OKFermeture

L'élève a ouvert un sachet, une caisse ou une armoire. Il déjà sélectionné certains des objets obtenus par cette erreur, mais voilà qu'il en désélectionne.	"Tu avais ouvert un sachet par erreur (comme les chiffres vont de 0 à 9, tu ne peux pas écrire le nombre si tu as plus de 9 bonbons). Sélectionne exactement 10 bonbons pour les regrouper en un sachet."

ag_TraiteRang

L'élève clique sur un objet qui n'appartient pas au rang courant.	"S'il y a plus de 9 caisses, sélectionnes-en EXACTEMENT 10 pour faire un report. Si le report est déjà fait ou s'il ne faut pas en faire, sélectionne toutes les caisses pour passer au rang suivant."

ag_PasEncoreRésultat

Le rang courant est R. L'élève a fait un report et veut cliquer sur l'objet obtenu avant d'avoir sélectionné tous les objets de rang R restants.	"Pour faire apparaître le résultat au rang des dizaines, sélectionne tous les sachets restants."

ag_PasEncoreRefermé

L'élève a ouvert par erreur un sachet, une caisse ou une armoire et veut continuer à traiter le rang courant.	"Dans le rectangle d'addition, tu as ouvert par erreur une armoire. Regroupe les caisses pour retrouver l'armoire."

ag_TropPourFermer

L'élève a sélectionné pour un regroupement plus de 10 objets.	"Pour regrouper des bonbons en un sachet, tu dois en sélectionner EXACTEMENT 10. Là, ça fait plus que 10."

Interaction entres les objets

Chaque élément visuel de la fenêtre se comporte comme si les autres n'existaient pas. Les seuls échanges de messages logiques se font :

- entre un objet et le gérant de l'exercice lorsque l'utilisateur manipule la souris;
- entre le gérant et un ou plusieurs objets lorsque l'action de l'élève a été analysée.

Gérant de l'exercice

Données

Identificateur	*Type*	*Sémantique*
BNombre	Array[1..2] of PBIcones	Les deux cardinaux à additionner
BSomme	PBIcones	Boîte pour effectuer l'addition en bonbons
BCalcul	PBAddSub	Boîte du calcul écrit
BExplic	PBTExte	Cadre pour afficher les instructions
BValeurs	PBouton	Bouton pour commencer une addition avec de nouvelles valeurs
RangCourant	VObjet	Ordre de grandeur sur lequel l'élève travaille
RangTermine	Array[VObjet] of Boolean	Pour chaque rang, vaut VRAI si toutes les actions nécessaires ont été faites (copie, report, sélection)
ToutAbaisse	Array[VObjet] of Boolean	Indique, pour chaque rang, si les objets des ensembles à additionner ont été copiés
ReportFait	Array[VObjet] of Boolean	Indique, pour chaque rang, si le report nécessaire a été effectué
OuvertErreur	Array[VObjet] of Boolean	Indique si un objet de tel ou tel rang a été ouvert par erreur
AddTerminee	Boolean	Indique si l'addition est terminée
IndicAction	Boolean	Vaut VRAI ssi il faut signaler explicitement qu'un report est nécessaire

Méthodes

`procedure Avertissement`

Objet : Afficher des explications dans une fenêtre de message.

Entrée :

ID	Integer	Identificateur de la chaîne de caractères à afficher

`procedure ChangeParamètres`

Objet : Déterminer la nouvelle valeur d'*IndicAction* après que l'utilisateur a modifié les options générales du programme.

`procedure ChoisitValeurs`

Objet : Selon les options, génénrer aléatoirement les deux nombres à additionner ou les demander à l'utilisateur.

Sortie :

Val1, Val2	Integer	Les deux valeurs à additionner
OK	Boolean	Dans le cas où l'utilisateur choisit lui-même les valeurs, OK vaut VRAI ssi l'utilisateur a validé la saisie.

`procedure ClickBouton`

Objet : Si le numéro de bouton donné en paramètre vaut 1, choisir deux nouvelles valeurs pour l'addition.

Entrée :

No	Byte	Numéro du bouton enfoncé

`procedure Commentaire`

Objet : Afficher les instructions dont l'identificateur est donné en paramètre

Entrée :

ID1, ID2	Integer	Identifiant de deux chaînes (contenues dans les ressources) à afficher

Remarque :

La procédure accepte deux paramètres. Si l'on souhaite ne lui passer que la référence d'une seule chaîne, la constante cg_Rien (-1) désignera la chaîne vide. Sinon, les deux chaînes désignées seront concaténées.

`procedure DoneExercice`

Objet : Signaler que l'addition est terminée et indiquer comment choisir de nouvelles valeurs;

Mettre à *VRAI* les variables *Stop* et *AddTerminée*.

`procedure DoneRang`

Objet :

Terminer les opérations sur le rang courant :

- mettre en noir (couleur du résultat) les objets correspondants du cadre d'addition;
- désélectionner la colonne correspondante du calcul écrit.

S'il n'y a plus rien à calculer

- **alors** terminer l'addition
- **sinon** passer au rang suivant.

`procedure GarnitMainWindow`

Objet : Placer dans la fenêtre principale du programme les éléments visuels de l'exercice (3 boîtes à icônes, 1 boîte de calcul écrit, 1 boîte de texte et 1 bouton).

`procedure InitExercice`

Objet : Initialiser l'addition avec deux valeurs

Entrée :

Param	Pointer	Inutilisé

`procedure InitRang`

Objet :

- Autoriser la sélection du rang courant dans les boîtes 1 et 2;
- Dans la boîte 3 (addition), ajouter le rang courant aux objets pouvant être ouverts et regroupés;
- Mettre en évidence le rang courant dans le calcul écrit;
- Choisir les instructions appropriées en fonction du cas de figure.
- S'il n'y a aucun objet de rang courant dans les boîtes 1, 2 et 3, passer au rang suivant.

`function Ouverture`

Objet : Déterminer quel pictogramme a été ouvert par erreur.

Sortie :

- le rang (VObjet) de l'objet le plus petit qui est encore ouvert;
- *rien* si aucun pictogramme n'est ouvert.

`procedure ReçoitMessage`

Objet :

- Déterminer le numéro de l'objet intiateur du message;
- Rediriger l'analyse du message vers une méthode spécialisée.

  Cette redirection est conforme à la typologie des erreurs envisagée supra (voir "Gestion des erreurs", p. 69)

Entrée :

Msg	TMsgDidact	Message envoyé par un des objets didactiques de la fenêtre

procedure Reçoit_MG_Sélect

Objet : Analyser une sélection opérée dans une des boîtes à icônes.

Boîtes 1 et 2

- Si tous les objets du rang courant sont sélectionnés, les copier dans la boîte d'addition; donner les instructions pour le traitement dans la boîte 3.
- Verrouiller ces objets pour qu'ils ne puissent être désélectionnés;

Boîte 3

- Si la boîte d'addition ne contient pas encore la copie de tous les objets du rang courant, le signaler;
- Sinon, si un regroupement ne doit plus être fait et que l'élève a sélectionné tous les objets restants du rang courant, compléter le calcul écrit et passer au rang suivant.

Entrée :

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

procedure Reçoit_MG_Désélect

Objet : Analyser une désélection opérée dans une des boîtes à icônes.

Boîtes 1 et 2

- Seuls les objets du rang courant peuvent être sélectionnés, et donc désélectionnés. Signaler que cette annulation est absurde.

Boîte 3

- Ici aussi, une désélection est absurde. Signaler comment traiter le rang courant.

Entrée :

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

procedure Reçoit_MG_Ouvre

Objet : Analyser l'ouverture d'un objet dans une des boîtes à icônes.

Si nous désignons par R le rang de l'objet ouvert ($1 < R \leq 4$), les opérations à effectuer sont :

- Enregistrer l'ouverture fautive pour le rang concerné R.
- Empêcher toute sélection dans les boîtes 1 et 2 au rang R.
- Dans le calcul écrit, placer un point d'interrogation comme résultat au rang Pred(R).
- Signaler que cette ouverture est absurde.

Entrée :

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

procedure Reçoit_MG_Ferme

Objet : Analyser le regroupement d'objets dans une des boîtes à icônes.

Soit R le rang des objets regroupés ($1 \leq R < 4$).

- Si R est le rang courant, afficher le report dans le calcul écrit et fournir les commentaires appropriés.
- Sinon, l'élève vient de regrouper des objets obtenus par une ouverture fautive.

Entrée :

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

procedure Erreur_MG_Sélect

Objet : Analyser une sélection refusée par une boîte à icônes.

Soit R le rang de l'objet concerné.

Boîtes 1 et 2

- R < RangCourant : expliquer que le rang R a déjà été traité.
- R > RangCourant : expliquer pourquoi il faut d'abord traiter le rang courant.

Boîte 3

- Si une ouverture n'a pas encore été corrigée, le signaler.
- Si l'élève clique dans le vide, voir si tous les objets à additionner ont été copiés. Dans ce case, le signales. Si tous les objets ont été copiés, indiquer comment traiter le rang.
- Si R < RangCourant, expliquer que le rang R a déjà été traité.
- Si R > RangCourant, expliquer pourquoi il faut d'abord traiter le rang courant.

Entrée :

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

procedure Erreur_MG_Désélect

Objet : Analyser une désélection refusée par une boîte à icônes.

Boîtes 1 et 2

- Réexpliquer comment copier les objets pour l'addition.

Boîte 3

- Si l'élève était en bonne voie de regrouper des objets obtenus par une ouverture fautive et qu'il les désélectionne, le remettre sur la voie.
- S'il n'y a pas d'objet ouvert, réexpliquer comment traiter le rang courant.

Entrée :

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

procedure Erreur_MG_Ouvre

Objet : Analyser une ouverture refusée par une des boîtes à icônes.

Soit R le rang de l'objet concerné.

Boîtes 1 et 2

- Indiquer pourquoi cette action est absurde

Boîte 3

- Si l'objet est un bonbon, signaler qu'on ne peut l'ouvrir

- Autrement, si R < RangCourant et que l'objet est de type résultat, indiquer que le rang a déjà été traité.
- Si R < RangCourant et que l'objet n'est pas de type résultat, il a été obtenu par ouverture, et cette action n'a pas encore été corrigée. Attirer l'attention sur ce fait.
- Si R ≥ RangCourant, expliquer pourquoi cette action est absurde.

Entrée : -

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

`procedure Erreur_MG_Ferme`

Objet : Analyser un regroupement refusé par une boîte à icônes.

Boîte 3

- Ce message n'a qu'une cause possible : l'élève a sélectionné plus de 10 objets. Expliquer pourquoi le report est refusé.

Entrée : -

IndOrig	Byte	Numéro de l'objet initiateur
Msg	TMsgDidact	Message

Fonction d'appoint

`FUNCTION DialogueValeurs`

Objet : exécuter un dialogue qui demande à l'utilisateur deux valeurs à additionner/soustraire. Dans le cas de l'addition, l'objet vérifie que les deux nombres ne sont pas nuls et que leur somme ne dépasse pas 9999.

Entrée :

Fenetre	PWindow	Fenêtre principale du programme
Op	Char	Opération qui sera effectuée sur les deux valeurs

Sortie :

Val1, Val2	Integer	Les deux valeurs choisies

La fonction renvoie VRAI si l'utilisateur a validé son choix, FAUX sinon

Améliorations possibles

Un bouton que l'élève enfonce quand il a sélectionné tous les objets du rang courant dans les deux nombres pictogrammes à additionner.

Un bouton que l'élève enfonce quand il a fini de traiter le rang courant dans la boîte d'addition.

POUR :

on évite que l'enfant n'effectue "par hasard" une action correcte;

si l'enfant ne sait pas comment procéder, il reçoit des explications sur le maniement de la souris.

CONTRE : cela surchage l'écran, qui comporte déjà six objets.

NEUTRE : dans la version actuelle, l'enfant obtient déjà des explications détaillées.

Le premier bouton n'apporte rien de plus si, dans les boîtes 1 et 2, l'élève

- tente de sélectionner des objets de différents ordres de grandeur;
- sélectionne une partie seulement des objets de l'ordre de grandeur courant, puis clique dans la boîte d'addition;
- sélectionne une partie des objets, puis clique sur un de ceux déjà sélectionnés;

Le second bouton n'apporte rien de plus si, dans la boîte 3, l'élève

- sélectionne un objet de rang inférieur au rang courant;
- clique dans le vide;
- clique sur l'objet de report du rang courant alors qu'il n'a pas encore tout abaissé.

Exercice 2

Description générale

L'élève effectue un calcul écrit dirigé. Au rang courant, il sélectionne les chiffres qui interviennent. Le résultat du rang est un nombre en 1 ou 2 chiffres, que l'élève doit replacer au bon endroit dans le calcul écrit. Une boîte à icônes montre l'évolution de la somme.

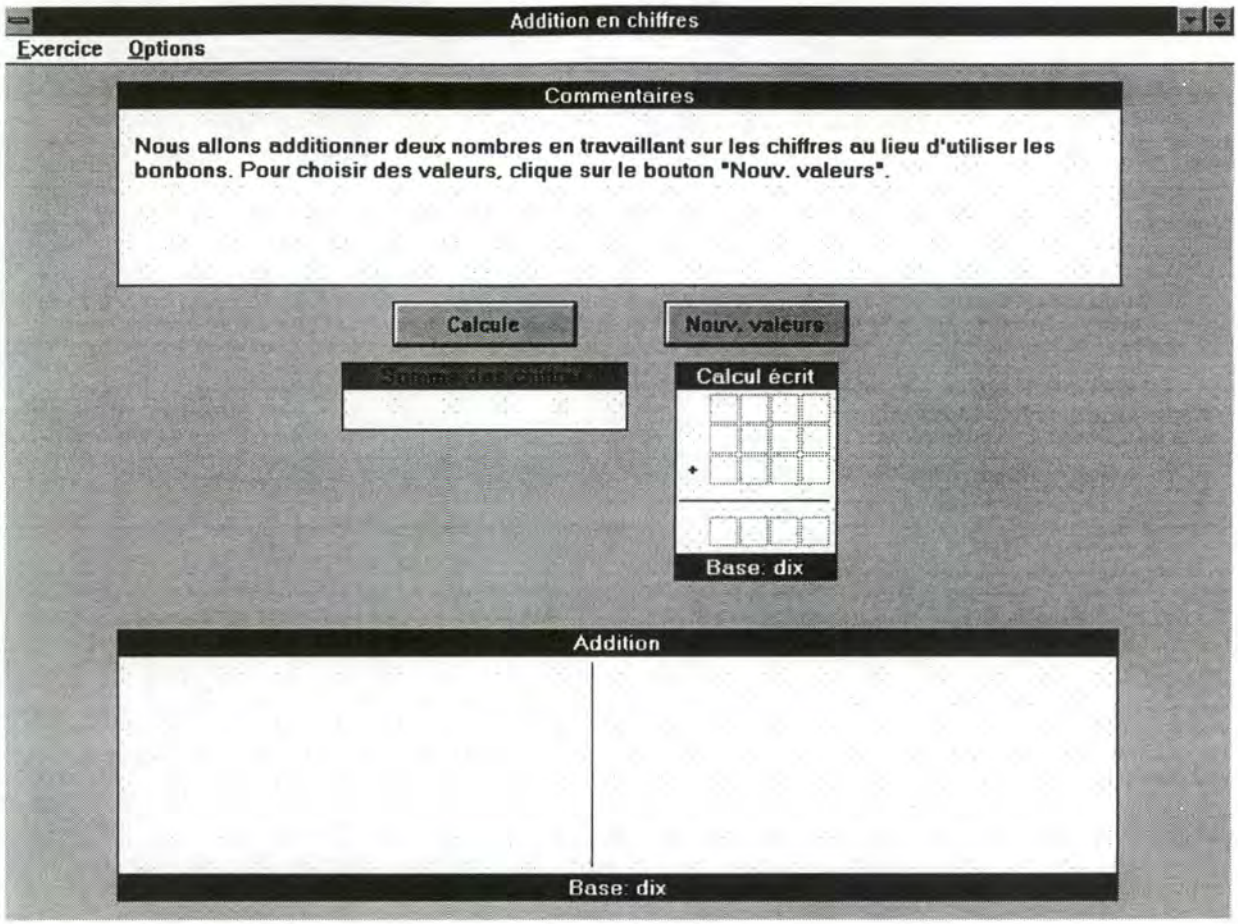
Description de l'écran

Boîtes

1. INSTRUCTIONS : boîte de texte dans laquelle s'inscrivent au fur et à mesure les explications sur le déroulement de l'exercice.
2. ADDITION : boîte à icônes qui montre l'évolution du résultat (copie des objets du rang courant, reports).
3. CALCUL ÉCRIT : l'addition sur laquelle l'élève travaille.
4. SOMME DES CHIFFRES : boîte destinée à contenir la somme des chiffres du rang courant.

Boutons

1. NOUV. VALEURS : bouton qui permet à l'élève de commencer une nouvelle addition.
2. CACLULE : à enfoncer pour inscrire la somme des chiffres du rang courant.



Déroulement de l'exercice

INITIALISATION

L'utilisateur clique sur le bouton "Nouv. valeurs" pour obtenir deux nombres. Ils sont affichés chiffres dans le calcul écrit.

BOUCLE

L'addition s'opère rang par rang, en commençant par les unités. Pour chaque ordre de grandeur, l'opération se déroule comme suit :

1. L'utilisateur sélectionne dans les deux nombres du calcul écrit tous les chiffres du rang courant R (il n'est pas nécessaire de sélectionner un 0).
2. Le programme ajoute une copie de ces chiffres à la boîte *Somme des chiffres*.
3. L'utilisateur clique sur le bouton Calcule. Selon l'option choisie, l'ordinateur calcule la somme des chiffres du rang courant ou la demande avant de l'inscrire.
4. Le résultat du rang courant comprend un ou deux chiffres. L'élève les place dans le calcul écrit par *drag-and-drop*.

TERMINAISON

L'addition est terminée lorsque :

- dans les deux nombres initiaux, tous les chiffres (≠0) de tous les rangs ont été sélectionnés;
- dans la boîte *Somme des chiffres*, pour chaque rang, le résultat a été replacé dans le calcul écrit.

Inscription des valeurs

Les valeurs à additionner sont déterminées aléatoirement par l'ordinateur ou demandées au pupitreur. La procédure de choix vérifie que la somme des nombres ne dépasse pas 9999. L'utilisateur peut choisir au préalable le nombre maximum de chiffres dans le menu *Options*. Si les valeurs sont choisies par l'ordinateur, elles doivent répondre à ce critère de taille.

Les résultats intermédiaires peuvent être donnés directement par l'ordinateur ou demandés à l'utilisateur lorsqu'il enfonce le bouton "Calcule". Parmi ces deux solutions, je ne pense pas qu'il en soit une meilleure que l'autre. Le seul but de cette option est d'insérer un pallier dans l'apprentissage.

Vue dans la perspective du long terme (les connaissances à posséder en bout de course), la première solution a le désagréable inconvénient d'éviter tout calcul à l'élève. Par contre, si l'ordinateur donne automatiquement la somme des chiffres du rang, l'enfant peut se concentrer exclusivement sur la sémantique du report et la structure du calcul écrit. Ainsi, il ne coupe pas son raisonnement, ne passe pas sans cesse du niveau logique au calcul mental.

Par la suite, dès que les tables d'addition sont connues, l'enfant peut donner sans effort la somme de deux ou trois chiffres. Le calcul des résultats intermédiaires n'est plus un obstacle. Et surtout, l'ordinateur teste à quel point les tables sont bien ou mal mémorisées.

Instructions données pendant l'exercice

DÉBUT ET FIN DE L'EXERCICE

`cg_Début`

Situation	*Texte*
Initialisation de l'exercice, avant même que l'élève ou l'ordinateur ne choisisse les deux premières valeurs.	"Nous allons additionner deux nombres en travaillant sur les chiffres au lieu d'utiliser les bonbons. Pour choisir des valeurs, clique sur le bouton *Nouv. valeurs*."

`cg_Fin`
`cg_ChoisisDesValeurs`

L'addition des deux valeurs est terminée.	"Voilà, l'addition est terminée. Pour choisir deux nombres et les additionner, clique sur le bouton <Valeurs>. \r\nEventuellement, consulte le menu Options | Générales | Qui choisit les valeurs ? ..."

INITIALISATION D'UN RANG

`cg_SélectCETout`

Le rang courant doit être traité.	"Dans le calcul écrit, sélectionne tous les chiffres au rang des unités. Quand c'est terminé, enfonce le bouton *Calcule* pour avoir la somme de ces chiffres."

OPÉRATIONS SUR LE RANG COURANT

`cg_CommentDéplacer`

L'utilisateur a copié les chiffres du rang courant et obtenu leur somme.	"Pour déplacer un chiffre de *Somme des chiffres* vers *Calcul écrit* : (1) clique sur le chiffre et laisse le bouton enfoncé, (2) déplace le chiffre, (3) relâche le bouton de la souris."

`cg_CopieASTout`

Idem.	"La somme des chiffres aux dizaines donne un nombre en 1 ou 2 chiffres : place-les au bon endroit dans le calcul écrit."

`cg_CopieASRésRep`

Idem. Il y 2 chiffres à replacer.	"La somme des chiffres aux dizaines donne un nombre en 2 chiffres : le premier chiffre est un report d'une centaine, le deuxième est le résultat au rang des dizaines. Place les deux chiffres dans le calcul écrit."

`cg_CopieASRésultat`

Idem. Il n'y a qu'un chiffre à replacer.	"La somme des chiffres aux dizaines donne un nombre en 1 chiffre : place-le comme résultat aux dizaines dans le calcul écrit."

Erreurs possibles

cg_ChoisisDesValeurs

Cause	*Réponse*
L'élève clique dans une des boîtes alors que l'addition est terminée.	"Pour choisir deux nombres et les additionner, clique sur le bouton <Nouv. valeurs>. Eventuellement, consulte le menu Options | Générales | Qui choisit les valeurs ? ..."

ag_Chiffre1PasSélect

Cause	*Exemple de réponse*
Alors qu'il n'a pas sélectionné le chiffre du rang courant dans le premier nombre, l'utilisateur soit enfonce le bouton calcule, soit clique sur un chiffre déjà sélectionné.	"Tu n'as pas sélectionné le chiffre des milliers du premier nombre. Il intervient aussi dans le calcul."

ag_Chiffre2PasSélect

Alors qu'il n'a pas sélectionné le chiffre du rang courant dans le second nombre, l'utilisateur soit enfonce le bouton calcule, soit clique sur un chiffre déjà sélectionné.	"Tu n'as pas sélectionné le chiffre des milliers du premier nombre. Il intervient aussi dans le calcul."

ag_ReportPasSélect

Alors qu'il n'a pas sélectionné le chiffre de report du rang courant, l'utilisateur soit enfonce le bouton calcule, soit clique sur un chiffre déjà sélectionné.	"N'oublie pas : le report au rang des dizaines compte aussi dans le résultat. Sélectionne-le..."

ag_PasDeReportCE

Dans le calcul écrit, l'utilisateur clique au-dessus du premier nombre alors qu'il n'y a pas de report au rang courant.	"Il n'y a pas de report aux milliers. Pas besoin de cliquer sur cette case."

ag_ChiffresCEDéjàCopiés

L'élève a déjà sélectionné tous les chiffres du rang courant et clique sur un chiffre déjà mis en évidence.	"Ce chiffre est déjà pris. Quand tu as sélectionné tous les chiffres des dizaines, clique sur le bouton *Calcule*"

ag_RangDéjàFait

Dans le calcul écrit, l'élève clique sur un chiffre du rang inférieur au rang courant.	"Tu as déjà calculé le résultat pour ce rang. Passe aux dizaines."

ag_RangPasEncore

Dans le calcul écrit, l'élève clique sur un chiffre du rang supérieur au rang courant.	"Suppose que tu calcules déjà le résultat aux dizaines. Si tu devais faire un report à ce rang, tu devrais changer le résultat."

ag_PasDeReportSA

L'élève a sélectionné tous les chiffres du rang courant qui interviennent et obtenu une somme en 1 chiffre. Dans la boîte "Somme des chiffres", il clique à gauche du seul chiffre à replacer.	"La somme des centaines fait moins que 10. Il n'y a pas de report à faire aux milliers."

ag_CaseInvalideSA

L'élève clique dans la boîte "Somme des chiffres", mais pas sur un chiffre à replacer.	"A gauche, tu as de 1 à 3 chiffres. A droite, tu as leur somme : un nombre entre 0 et 19. Les 1 ou 2 chiffres de cette somme, place-les comme résultat (et report) dans le calcul écrit."

ag_ReportVersRésultat

L'élève a prélevé dans la somme des chiffres le chiffre de report et l'a placé comme résultat dans le calcul écrit.	"Ce 1 veut dire que tu a transformé 10 bonbons en un sachet. Le sachet obtenu n'est pas encore le résultat : il faut l'ajouter aux autres qui existent déjà. Place le 1 comme report au rang des dizaines."

ag_RésultatVersReport

L'élève a prélevé dans la somme des chiffres le chiffre de résultat et l'a placé comme report dans le calcul écrit.	"Ce chiffre, c'est le nombre des bonbons qui restent après regroupement. Place-le comme résultat aux unités."

ag_ReportMauvaisRang

L'utilisateur a prélevé le chiffre de report et ne le place pas au bon rang dans le calcul écrit.	"Ce 1 veut dire que tu transformes 10 bonbons en un sachet : tu as 10 bonbons de moins, mais tu ajoutes un sachet. Place le 1 comme report au rang des dizaines."

ag_RésultatMauvaisRang

L'utilisateur a prélevé le chiffre de résultat et ne le place pas au bon rang dans le calcul écrit.	"Ce chiffre, c'est le nombre de sachets qui restent après regroupement. Place-le comme résultat aux dizaines."

ag_VersMauvaiseCase

L'utilisateur a prélevé un chiffre de la somme des chiffres et ne le replace pas comme résultat ni comme report.	"Place un chiffre comme résultat aux dizaines ou comme report aux centaines."

ag_EnfonceBoutonCalcule

L'élève a déjà copié tous les chiffres du rang courant mais n'a pas encore obtenu leur somme et clique dans boîte "Somme des chiffres" ou dans le calcu écrit.	"Quand tu as cliqué sur tous les chiffres au rang courant, enfonce le bouton *Calcule*. Il te donnera un ou deux chiffres à replacer comme résultat (et report) dans le calcul écrit."

Gérant de l'exercice

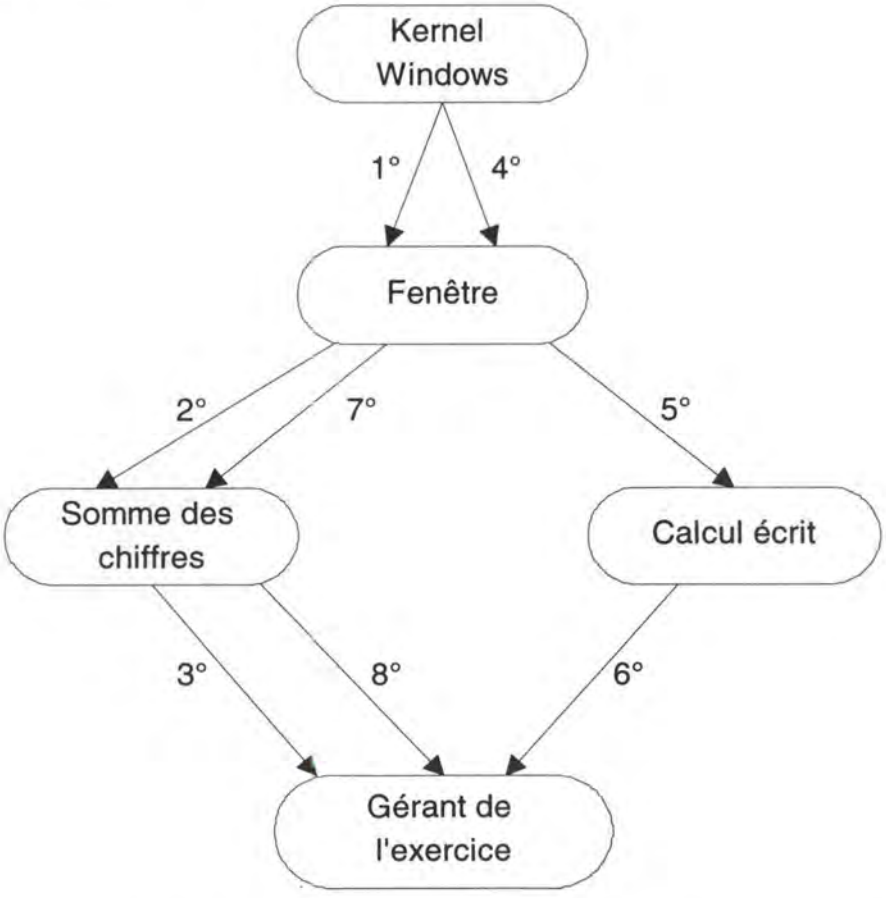
Données

Identificateur	*Type*	*Sémantique*
BCalcul	PBAddSub	Calcul écrit
BSimpleAdd	PBSimpleAdd	Boîte pour l'addition des chiffres du rang courant
BSomme	PBIcônes	Représentation pictographique de la somme
BExplic	PBTexte	Cadre pour afficher les instructions
BValeurs	PBouton	Permet de commencer une nouvelle addition
BChiffres	PBouton	Permet d'inscrire la somme des chiffres du rang courant
RangCourant	VObjet	Ordre de grandeur sur lequel l'élève travaille
NbChiffres	Byte	Nombre de chiffres copiés dans la boîte "Somme des chiffres"
RangTerminé	Array[VObjet] of Boolean	Pour chaque rang, vaut VRAI si toutes les actions nécessaires ont été faites (sélection, somme, replacement)
ToutCopié	Array[VObjet] of Boolean	Indique, pour chaque rang, si les chiffres à additionner ont été copiés
ReportFait	Array[VObjet] of Boolean	Indique, pour chaque rang, si le report nécessaire a été effectué
ADéplacer	Set of VCatégorie	Indique, pour chaque rang, les catégories de chiffres à copier de la boîte "Somme des chiffres" dans le calcul écrit.
Déplacés	Set of VCatégorie	Indique, pour chaque rang, les catégories des chiffres replacés dans la boîte "Somme des chiffres"
Déplacement	(RECORD)	Lorsque l'utilisateur a replacé un chiffre dans le calcul écrit, cette variable signale la catégorie du chiffre déplacé, le rang et la catégorie où il a été déposé.

PushCalcule	Boolean	Vaut VRAI ssi le bouton "Calcule" vient d'être enfoncé. Remis à FAUX après vérification des chiffres copiés.
AddTerminée	Boolean	Indique si l'addition est terminée
IndicAction	Boolean	Vaut VRAI ssi il faut signaler explicitement qu'un report est nécessaire

Réalisation du drag-and-drop

Nous pouvons schématiser comme suit la circulation des messages échangés entre les différents objets lorsque l'élève prélève un chiffre dans la boîte "Somme des chiffres" pour le déposer dans le calcul écrit :



1° L'utilisateur clique au-dessus d'une case de la boîte "Somme des chiffres". Le noyau de Windows crée un message *wm_lButtonDown*, qui est capturé par la fenêtre de l'application.

2° L'objet TFenêtreDidact analyse les coordonnées du click et identifie le cadre auquel le point visé correspond. La fenêtre appelle la méthode lButtonDown de la boîte "Somme des chiffres".

3° L'objet TBSimpleAdd détermine la case sélectionnée et le type de chiffre prélevé par l'utilisateur. La boîte place toutes les informations nécessaire dans un *TMsgDidact* les informations et le passe au gérant (*MessageGerant*) qui initialise la variable *Deplacement.*

4° L'utilisateur déplace le pointeur de la souris dans le calcul écrit, puis relâche le bouton. Le noyau génère un message *wm_lButtonUp.*

5° La fenêtre constate que le cadre où le bouton est relâché est différent de la boîte où l'utilisateur a cliqué. Le fenêtre envoie d'abord un message à la boîte de destination (le calcul écrit).

6° La boîte du calcul écrit signale au gérant à quel rang et à quelle ligne l'élève a relâché le bouton. Le gérant complète *Deplacement.*

7° La fenêtre signale de nouveau le relâchement du bouton, mais cette fois à la boîte "Somme des chiffres".

8° L'objet TBSimpleAdd indique au gérant de l'exercice que l'élève a relâché le bouton hors du cadre. Le programme peut analyser si le chiffre est correctement déplacé.

Note : l'ordre dans lequel la fenêtre avertit que l'utilisateur a relâché le bouton a son importance.

- La solution choisie ici garantit la cohérence du procédé. Le gérant peut déterminer si l'action est correcte dès qu'il reçoit le message de BSimpleAdd et ne doit lire aucune zone d'aucun objet ni analyser les coordonnées du click.
- Que se passe-t-il si l'ordre inverse est choisi (émission à BSimpleAdd, puis à BCalcul) ? Le gérant ne peut tirer de conclusions qu'après avoir reçu un message de BCalcul. Mais si l'utilisateur relâche le bouton dans un zone autre que le calcul écrit, le gérant n'en sait rien et ne peut donner d'explications à l'élève. Résoudre ce problème nécessitet soit de gonfler l'analyse avec des méthodes d'un niveau inférieur, soit de faire analyser par le gérant tout message wm_lButtonUp, ce qui est inacceptable.

Méthodes

procedure Avertissement

Objet : Afficher des explications dans une fenêtre de message.

Entrée :

ID	Integer	Identificateur de la chaîne de caractères à afficher

procedure ChangeParamètres

Objet : Déterminer la nouvelle valeur d'*IndicAction* après que l'utilisateur a modifié les options générales du programme.

procedure ChoisitValeurs

Objet : Selon les options, génénrer aléatoirement les deux nombres à additionner ou les demander à l'utilisateur.

Sortie :

Val1, Val2	Integer	Les deux valeurs à additionner
OK	Boolean	Dans le cas où l'utilisateur choisit lui-même les valeurs, OK vaut VRAI ssi l'utilisateur a validé la saisie.

procedure ClickBouton

Objet :

- Si le numéro de bouton donné en paramètre vaut 1, choisir deux nouvelles valeurs pour l'addition.
- Si le paramètre vaut deux, vérifier que les tous les chiffres du rang courant ont été sélectionnés.

Entrée :

No	Byte	Numéro du bouton enfoncé

procedure Commentaire

Objet : Afficher les instructions dont l'identificateur est donné en paramètre

Entrée :

ID1, ID2	Integer	Identifiant de deux chaînes (contenues dans les ressources) à afficher

Remarque :

La procédure accepte deux paramètres. Si l'on souhaite ne lui passer que la référence d'une seule chaîne, la constante cg_Rien (-1) désignera la chaîne vide. Sinon, les deux chaînes désignées seront concaténées.

`procedure DoneExercice`

Objet : Signaler que l'addition est terminée et indiquer comment choisir de nouvelles valeurs;

Mettre à *VRAI* la variable *AddTerminée*.

`procedure DoneRang`

Objet :

Terminer les opérations sur le rang courant :

- mettre en noir (couleur du résultat) les objets correspondants de la boîte à icônes;
- désélectionner la colonne correspondante du calcul écrit.

S'il n'y a plus rien à calculer

- **alors** terminer l'addition
- **sinon** passer au rang suivant.

`procedure GarnitMainWindow`

Objet : Placer dans la fenêtre principale du programme les éléments visuels de l'exercice.

`procedure InitExercice`

Objet : Initialiser l'addition avec deux valeurs

Entrée :

Param	Pointer	Inutilisé

`procedure InitRang`

Objet :

- Mettre en évidence le rang courant dans le calcul écrit;

- Autoriser la sélection du rang courant dans le calcul écrit;
- Vider la boîte "somme des chiffres";
- Choisir les instructions appropriées en fonction du cas de figure.

procedure ReçoitMessage

Objet :

- Déterminer le numéro de l'objet intiateur du message;
- Rediriger l'analyse du message vers une méthode spécialisée.

Entrée :

Msg	TMsgDidact	Message envoyé par un des objets didactiques de la fenêtre

procedure Erreur_MG_Désélect

Objet : donner des explications à l'utilisateur qui tente de désélectionner un chiffre du rang courant dans le calcul écrit.

Entrée :

IndOrig	Byte	Identificateur de la boîte initiatrice du message
Msg	TMsgDidact	Message adressé au gérant par la boîte "Calcul écrit"

procedure Erreur_MG_Prend

Objet : donner des explications à l'utilisateur, qui clique sur une mauvaise case de la boîte "Somme des chiffres".

Entrée :

IndOrig	Byte	Identificateur de la boîte initiatrice du message
Msg	TMsgDidact	Message adressé au gérant par la boîte "Somme des chiffres"

procedure Erreur_MG_Sélect

Objet : donner des explications à l'utilisateur, qui clique sur une mauvaise case du calcul écrit.

Entrée :

IndOrig	Byte	Identificateur de la boîte initiatrice du message
Msg	TMsgDidact	Message adressé au gérant par la boîte "Calcul écrit"

procedure ErreurGénérale

Objet :

- Si l'utilisateur a sélectionné tous les chiffres du rang courant et a inscrit leur somme, lui rappeler ce qu'il doit faire du (des) chiffre(s) obtenus.
- Si l'utilisateur a sélectionné tous les chiffres du rang courant mais n'a pas encore enfoncé le bouton "Calcule", lui dire de le faire.
- Si l'utilisateur n'a pas encore sélectionné tous les chiffres du rang courant, le lui rappeler.

procedure Reçoit_MG_Dépose

Objet :

- Enregistrer l'endroit du calcul écrit où l'élève a déposé un chiffre prélevé dans la boîte "Somme des chiffres".
- Si la case de destination choisie par l'élève est incorrecte, lui expliquer son erreur.

Entrée :

IndOrig	Byte	Identificateur de la boîte initiatrice du message
Msg	TMsgDidact	Message adressé au gérant par la boîte "Somme des chiffres"

procedure Reçoit_MG_Prend

Objet : enregistrer le type de chiffre (résultat, report) prélevé dans la boîte "Somme des chiffres".

Entrée :

IndOrig	Byte	Identificateur de la boîte initiatrice du message
Msg	TMsgDidact	Message adressé au gérant par la boîte "Somme des chiffres"

procedure Reçoit_MG_Sélect

Objet : enregistrer qu'un chiffre supplémentaire du rang courant est sélectionné dans le calcul écrit et en passer une copie à la boîte "Somme des chiffres".

Entrée :

IndOrig	Byte	Identificateur de la boîte initiatrice du message
Msg	TMsgDidact	Message adressé au gérant par la boîte "Calcul écrit"

procedure VérifieChiffresCE

Objet :

- Vérifier que, dans le calcul écrit, tous les chiffres (≠0) du rang courant sont sélectionnés.
- Si ce n'est pas le cas, le rappeler à l'utilisateur.

Fonction d'appoint

FUNCTION DialogueValeurs

Objet : exécuter un dialogue qui demande à l'utilisateur deux valeurs à additionner/soustraire. Dans le cas de l'addition, l'objet vérifie que les deux nombres ne sont pas nuls et que leur somme ne dépasse pas 9999.

Entrée :

Fenetre	PWindow	Fenêtre principale du programme
Op	Char	Opération qui sera effectuée sur les deux valeurs

Sortie :

Val1, Val2	Integer	Les deux valeurs choisies

La fonction renvoie VRAI si l'utilisateur a validé son choix, FAUX sinon

Améliorations possibles

Retours en arrière

Cet exercice met en oeuvre des notions d'un niveau plus élevé. Un élève pourrait l'aborder sans encore maîtriser à fond certains prérequis. Dans sa version actuelle, le gérant répond aux fautes identiques par des commentaires semblables. Si l'enfant répète une faute déjà commise, sans doute n'a-t-il pas compris les explications, ou a-t-il du mal à faire le lien avec ses connaissances actuelles.

Un didacticiel ramifié serait alors le bienvenu dans deux situations : les fautes répétées (il suffirait d'un historique tout simple, comme une table des fréquences) et les fautes graves.

La réponse du gérant pourrait être multiple :

- représenter l'erreur commise avec des bonbons;
- replacer l'élève dans un exercice spécifique pour les notions incomprises. Supposons que l'enfant additionne 347 et 681 et, arrivé aux dizaines, place mal le chiffre du report. L'ordinateur instancierait l'Exercice 1 avec les valeurs 347 et 681, effectuerait déjà les actions nécessaires pour le premier rang, et inviterait l'utilisateur à poursuivre à partir des sachets.

Répétitions

Prenons le cas où l'élève donne lui-même les résultats intermédiaires. Le gérant pourrait recenser les lignes mal connues des tables d'addition. De façon imprévisible, il afficherait de temps à autre une boîte de message du type : "N'oublie pas que 8 + 6 = 14 !". Cette initiative aiderait à mémoriser de deux façons :

- en stimulant la mémoire photographique;
- par un certain effet de surprise, l'élève aurait son attention mieux attirée.

Je crois cependant que pour être efficace, cette technique devrait se pratiquer avec parcimonie. En effet, si l'ordinateur interrompt trop souvent pour rappler les erreurs, l'utilisateur risque soit de ne même plus y prêter attention, soit de se décourager.

Faire le point

Une fois close la partie pratique de ce mémoire, deux questions fondamentales se posent. La première ressortit du présent : qu'apporte-t-il de neuf ? Il s'agit de faire le point sur les avantages immédiats que procure l'interface programmée. La seconde question porte sur l'avenir : quelles perspectives s'ouvrent à nous ?

Juger du présent

Le résultat est une bibliothèque d'objets programmés et de procédures qui simplifient la vie du programmeur de didacticiels.

D'une part, ce corpus est dédicacé à l'arithmétique. Il a pour philosophie sous-jacente une certaine représentation des nombres, conçue pour que l'enfant ne reçoive pas les notions abstraitres comme une création de l'esprit, mais plutôt qu'il ait l'intuition de la réalité qu'ils représentent. Un petit dessin vaut mieux qu'un long discours. L'élève qui travaille à l'écran comme sur les objets eux-mêmes perçoit l'opération comme une action bel et bien concrète. Il apprend à évaluer les ordres de grandeurs et réalise leur raison d'être.

De par sa complexité, la gestion du graphisme et de l'interface en Turbo Pascal sous Windows ne s'improvise pas. Ici, la solution apportée cache au programmeur toutes les finesses de l'affichage et des événements. Les fonctions API de Windows sont gérées au sein de modules dont le rédacteur d'exercices n'aura normalement pas à se soucier. L'interaction entre la fenêtre et les objets qu'elle contient se fait automatiquement. Sauf à réaliser certains effets spéciaux, le programme ne devra faire aucun appel explicite aux méthodes d'affichage.

Comme les objets gèrent eux-mêmes les éléments visuels associés, l'écriture d'exercices se recentre sur trois points principaux :

- la description de l'écran et l'initialisation des objets;
- l'enchaînement des séquences de l'exercice;
- le relevé des erreurs de l'élève, pour l'amener à se corriger.

Ces deux dernières procédures se trouvent facilitées, car les données transmises par les objets traduisent à un niveau conceptuel les événements élémentaires et les actions de l'utilisateur.

Ajoutons que le code est rédigé en programmation orientée objet, ce qui constitue un autre avantage. Par rapport aux solutions procédurales classiques, la POO :

- apporte des innovations (héritage et surcharge permettent de créer une hiérarchie de types avec un comportement peu ou prou différent);
- rend la sémantique plus transparente (la structure objet-méthode est plus proche de l'intuition);
- allège la syntaxe (les en-tête ont moins de paramètres).

Un regret toutefois. Par sa syntaxe, le Pascal se prête peu à rédiger des diagnostics d'erreurs. En nombre de lignes, cette analyse "gonfle" les exercices.

Présager de l'avenir

Comme exposé au chapitre "Elaborer l'outil", ce mémoire ne réalise pas un langage neuf mais prend une avancée dans cette direction. Aussi, ne commettons dons pas l'erreur de ne voir que les avantages immédiats. L'intérêt du travail réside aussi dans les perspectives qu'il offre.

Un champ élargi

L'outil conçu permet d'exposer la notion de nombre, l'addition et la soustraction. Il reste encore à traiter la multiplication et la division. Pour être bien assimilées, ces dernières opérations ont comme prérequis la compréhension des deux premières. Mais les objets créés ici ne permettent pas de bien présenter cette nouvelle matière.

Pour cela, il est nécessaire d'entreprendre une autre analyse pédagogique, d'imaginer de nouveaux exercices. C'est un passage obligé avant l'étude informatique et l'implémentation.

La démarche complète une fois appliquée aux quatre opérations, nous pourrions couvrir le programme d'arithmétique pour le cycle inférieur de l'enseignement primaire.

Un lien avec d'autres réalisations

La vie des écoliers serait simple si savoir calculer se limitait à connaître les quatre opérations. Hélas pour eux, l'étape suivante consiste à y greffer le calcul des expressions (numériques et algébriques). Les fractions ne sont pas les seules à causer des angoisses dans les rangs. Les élèves doivent aussi assimiler la priorité des opérations, la distributivité, les factorisations, etc.

Lorsqu'un enfant apprend peu à peu à manipuler les expressions, les quatre opérations et leurs propriétés sont censées connues. A ce stade, les explications à fournir concernent bien sûr la nouvelle matière, mais des points encore nébuleux dans les concepts de base causent aussi des erreurs dans la matière approfondie. Tout dépend comment s'est déroulé l'apprentissage préliminaire.

Dans ce cas, si le didacticiel est prévu pour les deux parties de la matière (opérations et expressions), si le diagnostic est bien affiné, l'enfant qui commet ce type d'erreur est replongé dans la matière correspondante. Après avoir revu les propriétés oubliées, il peut continuer l'exercice de départ.

Une utilisation plus pratique

Ces dernières années ont vu de fameux progrès dans l'interface des programmes pour PC. Une amélioration de qualité qui n'est pas gratuite, car les techniques de programmation mises en oeuvre se complexifient à l'avenant. Pour aider les concepteurs, différents outils ont vu le jour. Notamment pour dessiner les boîtes de dialogue.

Dans le même ordre d'idée, semblable utilitaire serait le bienvenu pour dessiner les écrans des exercices. Jusqu'à présent, il faut une petite dose de calcul mental et quelques essais pour obtenir quelque chose d'agréable. Deux solutions sont possibles et ont déjà fait leurs preuves.

La première idée est de traduire un schéma en une suite d'instructions. Le programmeur dessine à l'écran ce qu'il veut obtenir, et l'outil génère pour lui tout le code nécessaire.

Dans le second cas, le dessin est codé dans un langage de description reconnu par l'application. Cette épure est soit stockée dans un fichier de données séparé, soit intégrée au programme qui l'exploite, parmi les déclarations.

Un langage neuf

Le langage d'auteurs à part entière, objectif à long terme, devrait encore être défini. En se basant sur le fruit de ce mémoire, l'implémentation pourrait suivre deux voies distinctes.

La première consiste à rédiger un pré-compilateur, capable de traduire les instructions du langage en un programme Pascal qui utilise les librairies d'objets didactiques. Notons qu'écrire un compilateur direct reviendrait à jeter aux oubliettes le travail de programmation présenté ici.

La seconde est celle d'un interpréteur, écrit en Turbo Pascal et muni des mêmes libraires précitées, qui exécute les exercices en décodant leur description.

Le programme compilé l'emporterait en rapidité sur la version interprétée. Par contre, un interpréteur serait bien utile dans la phase de développement d'un exercice. Le créateur pourrait lancer plus vite son programme et l'interrompre pour lire les tables de valeurs. L'idéal serait d'utiliser un interpréteur pour les tests, et les compilateurs pour la version opérationnelle.

Un diagnostic plus lisible

Sans langage d'auteurs comme défini ci-dessus, le diagnostic d'erreurs est relativement lourd. De par sa syntaxe, le Pascal s'y prête mal. Des tests nombreux, en cascade, sont difficiles à rédiger et entravent la maintenance. Une solution est de confier comme d'ordinaire la gestion globale (déroulement de l'exercice) à des modules en Pascal, tandis que l'analyse des erreurs serait à charge de programmes écrits dans un langage d'intelligence artificielle sous Windows.

Ces considérations n'intéressent toutefois que le concepteur des outils. A partir du moment où le langage de haut niveau est défini, peu importe à l'enseignant comment il est implanté, ce sont les résultats qui comptent. Le système d'auteurs sera réputé bien conçu s'il permet de rédiger l'analyse des erreurs par un texte simple et concis.

Bibliographie

- BARON M. et allii, Environnements Interactifs d'Apprentissage avec Ordinateur, Paris, Eyrolles, 1993, 264 pp.
- BIERMAN D. et allii, Artificial Intelligence and Education, Amsterdam, IOS, 1989, 339 pp.
- BORLAND INTERNATIONAL, Turbo Pascal for Windows (Release 1.5, User Reference Manual), 1329 pp.
- GIROUX Y., Turbo Pascal pour Windows / Mode d'emploi, Paris, Sybex, 1991, 212 pp.
- GRIENENBERGER B., Dossier - Les logiciels éducatifs, in *icônes - Des souris et des hommes*, n° 41, juillet/août 1993, Roubaix.
- MINISTÈRE DE L'ÉDUCATION, Programme de mathématiques pour l'enseignement primaire (deux premiers cycles).
- SCHÖLLES R., Le grand livre de Turbo et Borland PASCAL 7.0, Paris, Micro Application, 1993, 1404 pp.
- SLEEMAN D. et allii, Intelligent Tutoring Systems, Paris, Academic Press, 1982, 345 pp.

Annexe - Code source

Le listing donné ici comprend tous les fichiers Pascal et de ressources Windows, à l'exception des icônes.

Ordre des fichiers

Cette page reprend le nom des différents fichiers, dans l'ordre de leur insertion dans cette annexe, avec une indication sur leur contenu.

Bibliothèque d'objets

FICHIER	*CONTENU*
Globaux.pas	Constantes, types et variables globaux
OBoite.pas	TBoite
OBTexte.pas	TBTexte
OBCases.pas	TBoiteCases
OBIcones	TBIcones
OCalcAS.pas	TCalcAddSub
OBCaptur.pas	TBoiteCapture
OBAddSub.pas	TBAddSub
DSimAdd.rc DSimAdd.pas	Boîte de dialogue pour TBSimAdd
OBSimAdd.pas	TBSimpleAdd
DDidact.rc DDidact.pas	Boîte de dialogue pour TDidacticiel
OBouton.pas	TBouton
ODidact.pas	TDidactVide TGerantVide TFenetreVide

Gérants d'exercices et programme principal

FICHIER	*CONTENU*
DValeurs.rc DValeurs.pas	Boîte de dialogue pour TGerantAdd001 et TGerantAdd002
SAdd001.rc	Chaînes de caractères pour TGerantAdd001
GAdd001.pas	TGerantAdd001
SAdd002.rc	Chaînes de caractères pour TGerantAdd002
GAdd002.pas	TGerantAdd002
Addition.pas	TDidacticiel

Globaux.pas

```
UNIT Globaux;

{-------------------------------------------------------------------------}
          INTERFACE
{-------------------------------------------------------------------------}

{$R IDidact2.RC}

USES WinTypes, WinProcs, OWindows, Strings, Validate;

CONST
{-------------------------------------------------------------------------
     Couleurs
-------------------------------------------------------------------------}
Blanc         = $00FFFFFF;
BlancCasse    = $00F8FFFF;
Bleu          = $00FF0000;
BleuFonce     = $00800000;
Brun          = $00008080;
Cyan          = $00FFFF00;
CyanFonce     = $00808000;
GrisClair     = $00C0C0C0;
GrisFonce     = $00808080;
Jaune         = $0000FFFF;
Magenta       = $00FF00FF;
MagentaFonce  = $00800080;
Noir          = $00000000;
Rouge         = $000000FF;
RougeFonce    = $00000080;
Vert          = $0000FF00;
VertFonce     = $00008000;
VertMoyen     = $00C0DCC0;

cr_fgTitreDefaut   = Blanc;
cr_bgTitreDefaut   = GrisClair;
cr_bgFondDefaut    = BlancCasse;

cr_fgTitreBIcones = Blanc;
cr_bgTitreBIcones = RougeFonce;
cr_bgFondBIcones  = BlancCasse;

cr_fgTitreBAddSub = Blanc;
cr_bgTitreBAddSub = VertFonce;
cr_bgFondBAddSub  = BlancCasse;

cr_fgTitreBTexte  = Blanc;
cr_bgTitreBTexte  = Bleu;
cr_bgFondBTexte   = BlancCasse;

cr_fgTitreBSimAdd = Noir;
cr_bgTitreBSimAdd = CyanFonce;
cr_bgFondBSimAdd  = BlancCasse;

cr_bgFondFenetre  = VertMoyen;

{-------------------------------------------------------------------------
     Identificateurs de message
-------------------------------------------------------------------------}
mg_Select     = $0001;
mg_Deselect   = $0002;
mg_Prend      = $0004;
mg_Depose     = $0008;
mg_Ferme      = $0010;
mg_Ouvre      = $0080;
mg_Reduit     = $0100;
mg_Termine    = $F000;

rr_SansErreur = $0000;   { l'opération a été effectuée }
rr_Select     = $0001;   { pas d'ouverture si un objet est sélectionné}
rr_MonoSelect = $0002;   { pas plus d'un type d'objet sélectionné à la fois }
rr_SelectPlus = $0004;   { pas de regroupement car plusieurs types sélectionnés }
rr_SelectTrop = $0008;   { le nombre d'objets sélectionnés dépasse la base }
```

```
rr_NoCase     = $000F;   { le case pointée ne peut être sélectionnée ou déplacée }
rr_Objet      = $0010;   { ce type d'objet ne peut être sélectionné }
rr_Genre      = $0020;   { ce genre d'objet ne peut être sélectionne }
rr_PasDePlace = $0100;   { pas assez de cases pour afficher }

{-------------------------------------------------------------------------------
     Paramètres des boîtes et fenêtres
-------------------------------------------------------------------------------}
BordIntBoite = 003;
{HauteurTitre = 016;}
BaseMax      = 016;
MaxComment   = 400;
LargeurIcone = 032;
HauteurIcone = 032;
EntreIcones  = 002;
MaxIcones    = 180;

cr_FondTitre  = GrisClair;
cr_EncreTitre = Blanc;


TYPE
{-------------------------------------------------------------------------------
    Types scalaires
-------------------------------------------------------------------------------}
VAffichage   = (af_Impossible, af_Cordes, af_Colonnes, af_Lignes,
                af_ContCord, af_ContCol, af_ContLig);
VObjet       = (Rien, Bonbon, Sachet, Caisse, Armoire);
VCategorie   = (Resultat, NNeg, ENeg, EPos, NPos, Report, Total);
VFlButton    = (fl_Sans, fl_Select, fl_Deselect, fl_Prend, fl_Deplace, fl_Depose);
VUsage       = (ub_Addition, ub_Soustraction, ub_Nombre, ub_Statique);
VNiveau      = (ni_Elementaire, ni_Moyen, ni_Avance);
VGuide       = (gd_Absent, gd_Bref, gd_Detaille);
VChoix       = (ch_Eleve, ch_Professeur, ch_Ordinateur);

{-------------------------------------------------------------------------------
     Types structurés
-------------------------------------------------------------------------------}
TRectangle = OBJECT
   X1, Y1,
   X2, Y2 : INTEGER;
   END;

TMsgDidact = RECORD
   MOrigine : POINTER;
   MMessage : Word;
   MAccepte : BOOLEAN;
   MNoCase  : INTEGER;
   MObjet   : VObjet;
   MGenre   : VCategorie;
   MCode    : Word;
   END;
TCaseBIcones   = RECORD
   Objet       : VObjet;
   Genre       : VCategorie;
   Select      : BOOLEAN;
   END;

PParametresPrg = ^TParametresPrg;
TParametresPrg = RECORD
   NBRangs    : Byte;
   Guide      : VGuide;
   Choix      : VChoix;
   Tables     : VChoix;
   END;

TNbObjets      = INTEGER;
TCommentaire   = ARRAY[0..MaxComment] OF CHAR;
TReprNombre    = ARRAY[Bonbon..Armoire] OF TNbObjets;
TDetailSelect  = ARRAY[VCategorie] OF Byte;
TCardinal      = ARRAY[VCategorie] OF TReprNombre;
T3Chiffres     = ARRAY[1..3] OF INTEGER;
PTableauIcones = ^TTableauIcones;
TTableauIcones = ARRAY[0..MaxIcones-1] OF TCaseBIcones;
TNbSelect      = ARRAY[VObjet, VCategorie] OF Byte;
TSetCategories = SET OF VCategorie;
```

```
   TSetByte       = SET OF Byte;
   TMiniChaine    = STRING[2];

   {-------------------------------------------------------------------------
        TRangeValidatorF : valide l'entrée de nombres
        Réécriture du message d'erreur en français
   -------------------------------------------------------------------------}

   PRangeValidatorF = ^TRangeValidatorF;
   TRangeValidatorF  = OBJECT (TRangeValidator)
      procedure Error; virtual;
      End;

   {-------------------------------------------------------------------------}
   CONST
   ObjetMax      = Armoire;
   ObjetDeRang   : ARRAY[0..4] OF VObjet  = (Rien, Bonbon, Sachet, Caisse, Armoire);
   RangDeObjet   : ARRAY[VObjet] OF Byte  = (0, 1, 2, 3, 4);
   NomObjet      : ARRAY[VObjet] OF PChar
                 = ('Rien','Bonbon','Sachet','Caisse','Armoire');
   NomGenre      : ARRAY[VCategorie] OF PChar
                 = ('Resultat','NNeg','ENeg','EPos','NPos','Report','Total');
   Chiffre       : ARRAY[0..BaseMax] OF CHAR
                 = '0123456789ABCDEF';
   ch_Interro    = 255;
   ch_Espace     = 254;

   {-------------------------------------------------------------------------}
   VAR
   HauteurTitre : Byte;
   StockIcone   : ARRAY [VObjet,VCategorie] OF HIcon;
   DessineTout  : TPaintStruct;
   SystemFont   : HFont;

   {-------------------------------------------------------------------------}

   FUNCTION  BitsA1 (var Adresse; Octets: Word): Word;
   PROCEDURE ChargeIcones;
   PROCEDURE ConvertitNombre (N: Integer; B: Byte; var T; C: Byte);
   PROCEDURE CreeBrosseSolide (var Adresse; Couleur: TColorRef);
   PROCEDURE CreePlumeSolide (var Adresse; Couleur: TColorRef);
   FUNCTION  Egalite (var A, B; Size: Word): Boolean;
   FUNCTION  EnToutesLettres (Valeur: LongInt): String;
   PROCEDURE MetAZero (var A; Taille: Integer);
   PROCEDURE Remplit (var A; Taille: Integer; Sequence: String);
   PROCEDURE TailleBouton (Fenetre: PWindow; Texte: PChar; var dX, dY: Integer);

   {-------------------------------------------------------------------------}
            IMPLEMENTATION
   {-------------------------------------------------------------------------}

+--FUNCTION BitsA1 (VAR Adresse; Octets: Word): Word;
|  VAR T : ARRAY[1..8192] OF Byte ABSOLUTE Adresse;
|      Resultat : Word;
|      Octet : Byte;
|      Ind : Word;
|      I : Byte;
+--BEGIN
|     Resultat := 0;
|  +--IF Octets < 8193 THEN BEGIN
|  |  +--FOR Ind := 1 TO Octets DO BEGIN
|  |  |      Octet := T[Ind];
|  |  |   +--FOR I := 0 TO 7 DO BEGIN
|  |  |   |      IF (Octet AND $01) = 1 THEN Inc(Resultat);
|  |  |   |      Octet := Octet SHR 1;
|  |  |   +--END;
|  |  +--END;
|  +--END;
|     BitsA1 := Resultat;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE ChargeIcones;
|  VAR Objet : VObjet;
|      Genre : VCategorie;
|      NomIcone : ARRAY[1..25] OF CHAR;
+--BEGIN
```

```
|     FOR Objet := Rien TO Armoire DO
|      +--FOR Genre := Resultat TO Report DO BEGIN
|      |      IF Objet = Rien
|      |      THEN StrLCopy (@NomIcone, NomObjet[Rien], 20)
|      |   +--ELSE BEGIN
|      |   |       StrLCopy (@NomIcone, NomObjet[Objet], 20);
|      |   |       StrLCat (@NomIcone, '_', 20);
|      |   |       StrLCat (@NomIcone, NomGenre[Genre], 20);
|      |   +-------END;
|      |      StockIcone[Objet, Genre] := LoadIcon (HInstance, @NomIcone);
|      +--END;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE ConvertitNombre (N: INTEGER; B: Byte; VAR T; C: Byte);
|  VAR Tableau : ARRAY[1..5] OF TNbObjets ABSOLUTE T;
|      Rang    : INTEGER;
+--BEGIN
|     IF C > 5 THEN C := 5;
|     MetAZero (Tableau, SizeOf(Tableau));
|     IF N < 0 THEN N := -N;
|  +--FOR Rang := 1 TO C DO BEGIN
|  |      Tableau[Rang] := N MOD B;
|  |      N := (N - Tableau[Rang]) DIV B;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE CreeBrosseSolide (VAR Adresse; Couleur: TColorRef);
|  VAR Brosse : HBrush ABSOLUTE Adresse;
+--BEGIN
|     IF Brosse <> 0 THEN DeleteObject (Brosse);
|     Brosse := CreateSolidBrush (Couleur);
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE CreePlumeSolide (VAR Adresse; Couleur: TColorRef);
|  VAR Plume : HPen ABSOLUTE Adresse;
+--BEGIN
|     IF Plume <> 0 THEN DeleteObject (Plume);
|     Plume := CreatePen (ps_Solid, 1, Couleur);
+--END;
   {-------------------------------------------------------------------------}
+--FUNCTION EnToutesLettres (Valeur: LongInt): STRING;
|  TYPE  TNom = STRING[12];
|  CONST NomUnites : ARRAY[0..19] OF TNom =
|           ('zéro','un','deux','trois','quatre',
|            'cinq','six','sept','huit','neuf',
|            'dix','onze','douze','treize','quatorze',
|            'quinze','seize','dix-sept','dix-huit','dix-neuf');
|        NomDizaines : ARRAY[1..9] OF TNom =
|           ('dix','vingt','trente','quarante','cinquante',
|            'soixante','septante','quatre-vingt','nonante');
|        NomRangs : ARRAY[0..5] OF TNom =
|           ('','mille','million','milliard','billion','billiard');
|        PlurielRangs : ARRAY[0..5] OF BOOLEAN =
|           (FALSE,FALSE,TRUE,TRUE,TRUE,TRUE);
|  VAR   Result    : STRING;
|        Chaine    : STRING;
|        SubChaine : STRING;
|        SubResult : STRING;
|        Unites, Dizaines, Centaines : Byte;
|        Nombre    : INTEGER;
|        Longueur  : Byte;
|        Rang      : Byte;
|        Dummy     : INTEGER;
|  +--PROCEDURE Ajoute (VAR Dest: STRING; Orig: STRING);
|  +--BEGIN
|  |     IF Dest = ''
|  |     THEN Dest := Orig
|  |     ELSE Dest := Dest + ' ' + Orig
|  +--END;
+--BEGIN
|     IF Valeur = 0 THEN EnToutesLettres := NomUnites[0] ELSE
|     IF Valeur < 0 THEN EnToutesLettres := 'moins '+EnToutesLettres(-Valeur)
|  +--ELSE BEGIN
|  |       Str (Valeur, Chaine);  Rang := 0;  Result := '';
|  |    +--WHILE NOT (Chaine = '') DO BEGIN
|  |    |     SubResult := '';
```

```
|  |  |     Longueur := Length (Chaine);
|  |  |     IF Longueur <= 3
|  |  |  +--THEN BEGIN
|  |  |  |        SubChaine := Chaine;
|  |  |  |        Chaine := ''
|  |  |  +-------END
|  |  |  +--ELSE BEGIN
|  |  |  |        SubChaine := Copy (Chaine, Longueur-2, 3);
|  |  |  |        Chaine[0] := Chr (Longueur - 3)
|  |  |  +-------END;
|  |  |     Val (SubChaine, Nombre, Dummy);
|  |  |  +--IF Nombre <> 0 THEN BEGIN
|  |  |  |     Centaines := Nombre DIV 100;
|  |  |  |     Nombre := Nombre - 100 * Centaines;
|  |  |  |     Dizaines := Nombre DIV 10;
|  |  |  |     Nombre := Nombre - 10 * Dizaines;
|  |  |  |     Unites := Nombre;
|  |  |  |  +--CASE Centaines OF
|  |  |  |  |  0    : BEGIN END;
|  |  |  |  |  1    : SubResult := 'cent';
|  |  |  |  |  2..9 : SubResult := NomUnites[Centaines] +' cent'
|  |  |  |  +--END;
|  |  |  |  +--CASE Dizaines OF
|  |  |  |  |+-00 : CASE Unites OF
|  |  |  |  ||       000  : IF Centaines > 1 THEN SubResult := SubResult + 's';
|  |  |  |  ||       001  : IF Rang <> 3
|  |  |  |  ||                THEN Ajoute (SubResult, NomUnites[Unites]);
|  |  |  |  ||       2..9 : Ajoute (SubResult, NomUnites[Unites]);
|  |  |  |  |+------END;
|  |  |  |  |  01 : Ajoute (SubResult, NomUnites[10+Unites]);
|  |  |  |  |+-08 : CASE Unites OF
|  |  |  |  ||       000  : Ajoute (SubResult, NomDizaines[Dizaines]+'s');
|  |  |  |  ||       1..9 : Ajoute (SubResult, NomDizaines[Dizaines]+'-'+NomUnites[Unites])
|  |  |  |  |+------END;
|  |  |  |  |+-ELSE CASE Unites OF
|  |  |  |  ||       000  : Ajoute (SubResult, NomDizaines[Dizaines]);
|  |  |  |  ||       001  : Ajoute (SubResult, NomDizaines[Dizaines] + '-et-un');
|  |  |  |  ||       2..9 : Ajoute (SubResult, NomDizaines[Dizaines] +'-'+ NomUnites[Unites])
|  |  |  |  |+------END
|  |  |  |  +--END;
|  |  |  |     IF Rang <> 0
|  |  |  |     THEN Ajoute (SubResult, NomRangs[Rang DIV 3]);
|  |  |  |     Val (SubChaine, Nombre, Dummy);
|  |  |  |     IF (Nombre > 1) AND PlurielRangs[Rang DIV 3]
|  |  |  |     THEN SubResult := SubResult + 's';
|  |  |  +--END;
|  |  |     IF Result = ''
|  |  |     THEN Result := SubResult
|  |  |  +--ELSE BEGIN
|  |  |  |        IF SubResult <> ''
|  |  |  |        THEN Result := SubResult +' '+ Result
|  |  |  +-------END;
|  |  |     Inc (Rang, 3);
|  |  +--END;
|  |     EnToutesLettres := Result;
|  +-------END;
+--END;
   {-------------------------------------------------------------------------}
+--FUNCTION  Egalite (VAR A, B; Size: Word): BOOLEAN;
|  VAR T1 : ARRAY[1..65500] OF Byte ABSOLUTE A;
|      T2 : ARRAY[1..65500] OF Byte ABSOLUTE B;
|      Ind : Word;
|      Result : BOOLEAN;
+--BEGIN
|     Result := TRUE;
|     Ind := 1;
|  +--WHILE Result AND (Ind <= Size) DO BEGIN
|  |     Result := T1[Ind] = T2[Ind];
|  |     Inc (Ind);
|  +--END;
|     Egalite := Result;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE MetAZero (VAR A; Taille: INTEGER);
|  VAR T : ARRAY[1..32767] OF Byte ABSOLUTE A;
|      I : INTEGER;
```

```
+--BEGIN
|     IF Taille > 0 THEN FOR I := 1 TO Taille DO T[I] := 0;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE Remplit (VAR A; Taille: INTEGER; Sequence: STRING);
|  VAR T : ARRAY[1..32767] OF Byte ABSOLUTE A;
|      I, J : INTEGER;
+--BEGIN
|  +--IF Taille > 0 THEN BEGIN
|  |     IF Sequence = '' THEN Sequence := #0;
|  |     IF Ord(Sequence[0]) > Taille THEN Sequence[0] := Chr(Taille);
|  |     I := 1;
|  |  +--WHILE I <= (Taille - Length(Sequence)) DO BEGIN
|  |  |     Move (Sequence[1], T[I], Length(Sequence));
|  |  |     Inc (I, Length(Sequence));
|  |  +--END;
|  |     J := 1;
|  |  +--WHILE I <= Taille DO BEGIN
|  |  |     Move (Sequence[J], T[I], 1);
|  |  |     Inc (I); Inc (J);
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE TailleBouton (Fenetre: PWindow; Texte: PChar; VAR dX, dY: INTEGER);
|  VAR DC : HDC;
|      Taille : LongInt;
+--BEGIN
|     DC := GetDC (Fenetre^.HWindow);
|     Taille := GetTextExtent (DC, Texte, StrLen(Texte));
|     dX := Lo(Taille) + 30;
|     dY := Hi(Taille) + 30;
|     ReleaseDC (Fenetre^.HWindow, DC);
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE CreeFonteSysteme (VAR AFont: HFont);
|  VAR ALogFont : TLogFont;
+--BEGIN
|    FillChar(ALogFont, SizeOf(TLogFont), #0);
|    WITH ALogFont DO
| +--BEGIN
| |    lfHeight         := HauteurTitre;
| |    lfWeight         := fw_SemiBold;
| |    lfItalic         := 0;
| |    lfUnderline      := 0;
| |    lfOutPrecision   := Out_Stroke_Precis;
| |    lfClipPrecision  := Clip_Stroke_Precis;
| |    lfQuality        := Default_Quality;
| |    lfPitchAndFamily:= Variable_Pitch;
| |    StrCopy(lfFaceName, 'MS Sans Serif');
| +--END;
|    AFont := CreateFontIndirect(ALogFont);
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE CreePaintStruct (VAR Info: TPaintStruct);
+--BEGIN
|  +--WITH Info DO BEGIN
|  |     HDC := 0;
|  |     fErase := TRUE;
|  |  +--WITH rcPaint DO BEGIN
|  |  |     Left  := 0;
|  |  |     Top   := 0;
|  |  |     Right := GetSystemMetrics (sm_cxFullScreen);
|  |  |     Bottom:= GetSystemMetrics (sm_cyFullScreen);
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE InitHauteurTitre;
+--BEGIN
|     HauteurTitre := Round (GetSystemMetrics (sm_cyCaption)*8/9);
+--END;

   {-------------------------------------------------------------------------
        TRangeValidatorF
   -------------------------------------------------------------------------}
```

```
+--PROCEDURE TRangeValidatorF.Error;
|  VAR Comment : TCommentaire;
|      MinMax  : ARRAY[1..2] OF INTEGER;
+--BEGIN
|     MinMax[1] := Min;  MinMax[2] := Max;
|     WVSPrintF (@Comment, 'S''il te plaît, entre un nombre compris entre %i et %i...',
|        MinMax);
|     MessageBox (GetActiveWindow, @Comment, 'Validation',
|        mb_IconInformation OR mb_OK);
+--END;

   {-------------------------------------------------------------------------
        INITIALISATION
   -------------------------------------------------------------------------}
   BEGIN
      ChargeIcones;
      InitHauteurTitre;
      CreePaintStruct (DessineTout);
      CreeFonteSysteme (SystemFont);
   END.
```

OBoite.pas

```
UNIT OBoite;

{-------------------------------------------------------------------------}
        INTERFACE
{-------------------------------------------------------------------------}

USES Objects, OWindows, WinTypes, WinProcs, Strings,
     Globaux;

TYPE
{-------------------------------------------------------------------------
        TBoite
-------------------------------------------------------------------------}

PBoite = ^TBoite;
TBoite = OBJECT (TRectangle)
   Fenetre      : PWindow;
   Peinte       : Boolean;
   Active       : Boolean;
   Bord         : TRectangle;
   Titre        : PChar;
   SousTitre    : PChar;
   OKSousTitre  : Boolean;
   EncreTitre   : TColorRef;
   FondTitre    : TColorRef;
   FondBoite    : TColorRef;
   Plume        : HPen;
   Effaceur     : HPen;
   BrosseFond   : HBrush;
   BrosseTitre  : HBrush;
   FlButton     : VFlButton;

   constructor Init             (UnParent: PWindow; UnTitre: PChar; TitreBis: Boolean;
                                 AX, AY, BX, BY: Integer);
   destructor  Done;            virtual;

   procedure   AfficheContenu   (DC: HDC; Info: TPaintStruct); virtual;
   procedure   AfficheSousTitre (DC: HDC);
   procedure   AfficheTitre     (DC: HDC);
   procedure   Cache            (DC: HDC);
   function    Contient         (Msg: TMessage): Boolean;
   procedure   EffaceContenu    (DC: HDC); virtual;
   procedure   LacheDC           (var DC: HDC);
   procedure   Montre           (DC: HDC; Info: TPaintStruct);
   procedure   PrendDC          (var DC: HDC);
   procedure   PrendSousTitre   (UnSousTitre: PChar);

   procedure   lButtonDown    (var Msg: TMessage);    virtual;
   procedure   lButtonDblClk  (var Msg: TMessage);    virtual;
   procedure   lButtonUp      (var Msg: TMessage);    virtual;
   procedure   lButtonShift   (var Msg: TMessage);    virtual;
   procedure   MouseMove      (var Msg: TMessage);    virtual;
   procedure   rButtonDown    (var Msg: TMessage);    virtual;
   procedure   rButtonUp      (var Msg: TMessage);    virtual;
   END;

{-------------------------------------------------------------------------}
        IMPLEMENTATION
{-------------------------------------------------------------------------}
{-------------------------------------------------------------------------
        TBoite
-------------------------------------------------------------------------}

CONSTRUCTOR TBoite.Init (UnParent: PWindow; UnTitre: PChar; TitreBis: BOOLEAN;
           AX, AY, BX, BY: INTEGER);
BEGIN
   Fenetre := UnParent;
   Titre := UnTitre;
   SousTitre := NIL;
   OKSousTitre := TitreBis;

   X1 := AX;  Y1 := AY;  X2 := BX;  Y2 := BY;
   Bord.X1 := BordIntBoite;
```

```
      Bord.Y1 := HauteurTitre + BordIntBoite;
      Bord.X2 := BordIntBoite;
      IF TitreBis THEN Bord.Y2 := BordIntBoite + HauteurTitre ELSE Bord.Y2 := BordIntBoite;

      FlButton := fl_Sans;
      Peinte := FALSE;
      Active := TRUE;
      EncreTitre := cr_fgTitreDefaut;
      FondTitre := cr_bgTitreDefaut;
      FondBoite := cr_bgFondDefaut;
      BrosseTitre := 0;  CreeBrosseSolide (BrosseTitre, FondTitre);
      BrosseFond := 0;  CreeBrosseSolide (BrosseFond, FondBoite);
      Plume := 0; CreePlumeSolide (Plume, Noir);
      Effaceur := 0; CreePlumeSolide (Effaceur, FondBoite);
   END;
   {-------------------------------------------------------------------------}
   DESTRUCTOR TBoite.Done;
+--BEGIN
|     IF SousTitre <> NIL THEN StrDispose (SousTitre);
|     DeleteObject (Plume);
|     DeleteObject (Effaceur);
|     DeleteObject (BrosseTitre);
|     DeleteObject (BrosseFond);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.AfficheContenu (DC: HDC; Info: TPaintStruct);
+--BEGIN
|+-IF Peinte THEN BEGIN
||    SelectObject (DC, BrosseFond);
||    SelectObject (DC, Effaceur);
||    Rectangle (DC, X1 + Bord.X1, Y1 + Bord.Y1, X2 - Bord.X2, Y2 - Bord.Y2);
|+-END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.AfficheSousTitre (DC: HDC);
   VAR Rect: TRect;
+--BEGIN
|+-IF Peinte AND OKSousTitre THEN BEGIN
|| +--WITH Rect DO BEGIN
|| |     Left   := X1;
|| |     Right  := X2;
|| |     Top    := Y2 - HauteurTitre;
|| |     Bottom := Y2;
|| +--END;
||    SelectObject (DC, SystemFont);
||    SelectObject (DC, Plume);
||    SelectObject (DC, BrosseTitre);
||    Rectangle (DC, Rect.Left, Rect.Top, Rect.Right, Rect.Bottom);
||
|| +--IF SousTitre <> NIL THEN BEGIN
|| |     Inc (Rect.Top, 1);  Dec (Rect.Bottom, 1);
|| |     SetTextColor (DC, EncreTitre);  SetBkColor (DC, FondTitre);
|| |     DrawText (DC, SousTitre, StrLen(SousTitre), Rect,
|| |        DT_Center OR DT_VCenter OR DT_SingleLine);
|| +--END;
|+-END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.AfficheTitre (DC: HDC);
   VAR Rect: TRect;
+--BEGIN
|+-IF Peinte THEN BEGIN
|| +--WITH Rect DO BEGIN
|| |     Left   := X1;
|| |     Right  := X2;
|| |     Top    := Y1;
|| |     Bottom := Y1 + HauteurTitre;
|| +--END;
||    SelectObject (DC, SystemFont);
||    SelectObject (DC, Plume);
||    SelectObject (DC, BrosseTitre);
||    Rectangle (DC, Rect.Left, Rect.Top, Rect.Right, Rect.Bottom);
||
|| +--IF Titre <> NIL THEN BEGIN
|| |     Inc (Rect.Top, 1);  Dec (Rect.Bottom, 1);
|| |     SetTextColor (DC, EncreTitre);  SetBkColor (DC, FondTitre);
```

```
|| |     DrawText (DC, Titre, StrLen(Titre), Rect, DT_Center OR DT_VCenter OR DT_SingleLine);
|| +--END;
|+-END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.Cache (DC: HDC);
+--BEGIN
|     Peinte := FALSE;
|     SelectObject (DC, BrosseFond);
|     SelectObject (DC, Effaceur);
|     Rectangle (DC, X1, Y1, X2, Y2);
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBoite.Contient (Msg: TMessage): BOOLEAN;
   VAR X, Y : INTEGER;
+--BEGIN
|     X := Msg.lParamLo;
|     Y := Msg.lParamHi;
|     Contient := (X > X1) AND (X < X2) AND (Y > Y1) AND (Y < Y2);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.EffaceContenu (DC: HDC);
+--BEGIN
|     SelectObject (DC, BrosseFond);
|     SelectObject (DC, Effaceur);
|     Rectangle (DC,
|        X1 + Bord.X1, Y1 + Bord.Y1,
|        X2 - Bord.X2, Y2 - Bord.Y2);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.Montre (DC: HDC; Info: TPaintStruct);
   VAR Rect    : TRect;
+--BEGIN
|     Peinte := TRUE;
|     SelectObject (DC, Plume);
|     SelectObject (DC, BrosseFond);
|     Rectangle (DC, X1, Y1, X2, Y2);
|
|     AfficheTitre (DC);
|     AfficheContenu (DC, Info);
|     AfficheSousTitre (DC);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.PrendDC (VAR DC: HDC);
   VAR DC_Aux : HDC;
+--BEGIN
|     DC_Aux := GetDC (Fenetre^.HWindow);
|     IF DC_Aux <> 0 THEN DC := DC_Aux;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.PrendSousTitre (UnSousTitre: PChar);
   VAR UnDC : HDC;
+--BEGIN
|     IF SousTitre <> NIL THEN StrDispose (SousTitre);
|     SousTitre := StrNew (UnSousTitre);
|     UnDC := GetDC (Fenetre^.HWindow);
|     AfficheSousTitre (UnDC);
|     ReleaseDC (Fenetre^.HWindow, UnDC);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.LacheDC (VAR DC: HDC);
+--BEGIN
|  +--IF DC <> 0 THEN BEGIN
|  |     ReleaseDC (Fenetre^.HWindow, DC);
|  |     DC := 0;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoite.lButtonDown (VAR Msg: TMessage);
   BEGIN END;

   PROCEDURE TBoite.lButtonDblClk (VAR Msg: TMessage);
   BEGIN END;

   PROCEDURE TBoite.lButtonUp (VAR Msg: TMessage);
   BEGIN END;
```

```
PROCEDURE TBoite.lButtonShift (VAR Msg: TMessage);
BEGIN END;

PROCEDURE TBoite.MouseMove (VAR Msg: TMessage);
BEGIN END;

PROCEDURE TBoite.rButtonDown (VAR Msg: TMessage);
BEGIN END;

PROCEDURE TBoite.rButtonUp (VAR Msg: TMessage);
BEGIN END;

{------------------------------------------------------------------------------
         INITIALISATION
------------------------------------------------------------------------------}
END.
```

OBTexte.pas

```
UNIT OBTexte;

{-------------------------------------------------------------------------}
          INTERFACE
{-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, Strings,
     Globaux, OBoite, OBouton;

TYPE
{-------------------------------------------------------------------------}
     Objet TBTexte
{-------------------------------------------------------------------------}

PBTexte = ^TBTexte;
TBTexte = OBJECT (TBoite)
   EncreTexte  : TColorRef;
   Texte       : Array[1..MaxComment] of Char;
   constructor Init              (UnParent: PWindow; UnTitre: PChar;
                                  AX, AY, BX, BY: Integer);
   procedure   AfficheContenu    (DC: HDC; Info: TPaintStruct); virtual;
   procedure   PrendTexte        (UnTexte: TCommentaire);
   procedure   VideContenu;
   END;

{-------------------------------------------------------------------------}
          IMPLEMENTATION
{-------------------------------------------------------------------------}

CONSTRUCTOR TBTexte.Init
            (UnParent: PWindow; UnTitre: PChar; AX, AY, BX, BY: INTEGER);
BEGIN
   TBoite.Init (UnParent, UnTitre, FALSE, AX, AY, BX, BY);
   Bord.X1 := 10;  Bord.Y1 := HauteurTitre + 15;  Bord.X2 := 10;  Bord.Y2 := 10;
   EncreTitre := cr_fgTitreBTexte;
   FondTitre  := cr_bgTitreBTexte;
   FondBoite  := cr_bgFondBTexte;
   IF cr_bgFondBTexte <> cr_bgFondDefaut
   THEN CreeBrosseSolide (BrosseFond, cr_bgFondBTexte);
   IF cr_bgTitreBTexte <> cr_bgTitreDefaut
   THEN CreeBrosseSolide (BrosseTitre, cr_bgTitreBTexte);
   EncreTexte := Noir;
   VideContenu;
END;
{-------------------------------------------------------------------------}
PROCEDURE TBTexte.AfficheContenu (DC: HDC; Info: TPaintStruct);
VAR Rect : TRect;
+--BEGIN
|     TBoite.EffaceContenu (DC);
|     SelectObject (DC, SystemFont);
|  +--WITH Rect DO BEGIN
|  |     Left   := X1 + Bord.X1 + 1;
|  |     Right  := X2 - Bord.X2 - 1;
|  |     Top    := Y1 + Bord.Y1 + 1;
|  |     Bottom := Y2 - Bord.Y2 - 1;
|  +--END;
|     SetTextColor (DC, EncreTexte);  SetBkColor (DC, FondBoite);
|     DrawText (DC, @Texte, StrLen(@Texte), Rect,
|               DT_Left OR DT_WordBreak);
+--END;
{-------------------------------------------------------------------------}
PROCEDURE TBTexte.PrendTexte (UnTexte: TCommentaire);
VAR UnDC : HDC;
+--BEGIN
|     MetAZero (Texte, SizeOf(Texte));
|     StrLCopy (@Texte, @UnTexte, MaxComment);
|  +--IF Peinte THEN BEGIN
|  |     PrendDC (UnDC);
|  |     AfficheContenu (UnDC, DessineTout);
|  |     LacheDC (UnDC);
|  +--END;
+--END;
{-------------------------------------------------------------------------}
```

```
   PROCEDURE TBTexte.VideContenu;
+--BEGIN
|     MetAZero (Texte, SizeOf(Texte));
+--END;

   {-------------------------------------------------------------------------
        INITIALISATION
   -------------------------------------------------------------------------}
   END.
```

OBCases.pas

```
UNIT OBCases;

{-------------------------------------------------------------------------}
        INTERFACE
{-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, Strings,
     Globaux, OBoite, ODidact;

TYPE
{-------------------------------------------------------------------------
     Objet TBoiteCases
-------------------------------------------------------------------------}

PBoiteCases = ^TBoiteCases;
TBoiteCases = OBJECT (TBoite)
   DimCase     : TPoint;
   NbCases     : TPoint;
   PlumeCase   : HPen;
   Retrait     : TRectangle;
   constructor Init            (UnParent: PWindow; UnTitre: PChar;
               TitreBis: Boolean; AX, AY: Integer; NbX, NbY, DimX, DimY: Byte);
   destructor  Done;           virtual;
   function    CasePointee     (var Msg: TMessage) : Integer;
   procedure   CoordCase       (NoCase: Integer; var Rect: TRect);
   procedure   SelectionneCase (DC: HDC; NoCase: Integer; Oui: Boolean); virtual;
   function    TotalCases:     Integer;
   End;

{-------------------------------------------------------------------------}
        IMPLEMENTATION
{-------------------------------------------------------------------------}

CONSTRUCTOR TBoiteCases.Init (UnParent: PWindow; UnTitre: PChar; TitreBis: BOOLEAN;
            AX, AY: INTEGER; NbX, NbY, DimX, DimY: Byte);
VAR BX, BY : INTEGER;
BEGIN
   Bord.X1 := BordIntBoite;
   Bord.Y1 := BordIntBoite + HauteurTitre;
   Bord.X2 := BordIntBoite;
   IF TitreBis THEN Bord.Y2 := BordIntBoite + HauteurTitre ELSE Bord.Y2 := BordIntBoite;
   DimCase.X := DimX;
   DimCase.Y := DimY;
   NbCases.X := NbX;
   NbCases.Y := NbY;
   BX := AX + Bord.X1 + DimCase.X * NbCases.X + Bord.X2 + 1;
   BY := AY + Bord.Y1 + DimCase.Y * NbCases.Y + Bord.Y2 + 1;
   Retrait.X1 := 0;
   Retrait.Y1 := 0;
   Retrait.X2 := 1;
   Retrait.Y2 := 1;
   TBoite.Init (UnParent, UnTitre, TitreBis, AX, AY, BX, BY);
   PlumeCase    := CreatePen (ps_Solid, 1, GrisClair);
END;
{-------------------------------------------------------------------------}
DESTRUCTOR TBoiteCases.Done;
+--BEGIN
|    inherited Done;
|    DeleteObject (PlumeCase);
+--END;
{-------------------------------------------------------------------------}
FUNCTION TBoiteCases.CasePointee (VAR Msg: TMessage) : INTEGER;
VAR No, NoX, NoY : INTEGER;
    X, Y : INTEGER;
+--BEGIN
|    X := Msg.lParamLo;
|    Y := Msg.lParamHi;
|    IF (X > (X1+Bord.X1)) AND (X < (X2-Bord.X2-1)) AND
|       (Y > (Y1+Bord.Y1)) AND (Y < (Y2-Bord.Y2-1))
|  +--THEN BEGIN
|  |       NoX := (X - X1 - Bord.X1) DIV DimCase.X;
|  |       NoY := (Y - Y1 - Bord.Y1) DIV DimCase.Y;
|  |       No := NoY * NbCases.X  +  NoX;
```

```
|  |       IF No < TotalCases
|  |       THEN CasePointee := No
|  |       ELSE CasePointee := -1;
|  +-------END
|     ELSE CasePointee := -1;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoiteCases.CoordCase (NoCase: INTEGER; VAR Rect: TRect);
   VAR Li, Co : INTEGER;
+--BEGIN
|+-WITH Rect DO BEGIN
||    Li := NoCase DIV NbCases.X;
||    Co := NoCase MOD NbCases.X;
||    Left  := X1 + (DimCase.X * Co) + Bord.X1;
||    Top   := Y1 + (DimCase.Y * Li) + Bord.Y1;
||    Right := Left + DimCase.X - 1;
||    Bottom:= Top + DimCase.Y - 1;
|+-END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoiteCases.SelectionneCase (DC: HDC; NoCase: INTEGER; Oui: BOOLEAN);
   VAR Rect : TRect;
+--BEGIN
|  +--IF Peinte AND (NoCase < TotalCases) THEN BEGIN
|  |     CoordCase (NoCase, Rect);
|  |  +--WITH Rect DO BEGIN
|  |  |     Inc (Left, Retrait.X1);  Dec (Right, Retrait.X2);
|  |  |     Inc (Top, Retrait.Y1);   Dec (Bottom, Retrait.Y2);
|  |  |     IF Oui
|  |  |     THEN SelectObject (DC, PlumeCase)
|  |  |     ELSE SelectObject (DC, Effaceur);
|  |  |     MoveTo (DC, Left, Top);
|  |  |     LineTo (DC, Right, Top);    LineTo (DC, Right, Bottom);
|  |  |     LineTo (DC, Left, Bottom);  LineTo (DC, Left, Top);
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBoiteCases.TotalCases : INTEGER;
+--BEGIN
|     TotalCases := NbCases.X * NbCases.Y;
+--END;
   {-------------------------------------------------------------------------
            INITIALISATION
   -------------------------------------------------------------------------}
   END.
```

OBIcones.pas

```
UNIT OBIcones;

{-------------------------------------------------------------------------}
        INTERFACE
{-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, Strings,
     Globaux, OBCases, ODidact;

TYPE
{-------------------------------------------------------------------------
     Objet TBIcones
-------------------------------------------------------------------------}

PBIcones    = ^TBIcones;
TBIcones    = OBJECT (TBoiteCases)
   Inter        : TPoint;
   Card         : TCardinal;
   Base         : Byte;
   Contenu      : PTableauIcones;
   AncContenu   : PTableauIcones;
   MonoSelect   : Boolean;
   NbSelect     : TNbSelect;
   Orientation  : VAffichage;
   PlumeLimite  : HPen;
   LigneLimite  : TRect;
   OKLimite     : Boolean;
   OKSelect     : Set of VObjet;
   OKDeselect   : Set of VObjet;
   OKOuvrir     : Set of VObjet;
   OKFermer     : Set of VObjet;
   SiOuvrir     : Array[VCategorie] of VCategorie;
   SiFermer     : Array[VCategorie] of VCategorie;

   constructor Init                (UnParent: PWindow; UnTitre: PChar;
                                    AX, AY, NbX, NbY : Integer);
   destructor  Done;               virtual;

   procedure   AfficheContenu      (DC: HDC; Info: TPaintStruct); virtual;
   function    AffichageOK         (N: TReprNombre) : VAffichage;
   procedure   AfficheCase         (DC: HDC; NoCase: Integer);
   function    CumulSelect         (UnObjet: VObjet; Genres: TSetCategories): Word;
   procedure   DetailleSelection   (UnObjet: VObjet; var Detail: TDetailSelect);
   function    LargeurSelection:   Byte;
   procedure   MessageGerant       (ID: Word; UnObjet: VObjet; UnGenre: VCategorie;
                                   Code: Word); virtual;
   function    PrendValeur         (Val: Integer; BaseNum: Byte): Boolean;
   function    PrendCardinal       (N: TCardinal; BaseNum: Byte): Boolean;
   procedure   RegroupeSelection;
   procedure   SelectionneCase     (DC: HDC; NoCase: Integer; Oui: Boolean); virtual;
   procedure   SelectionneObjet    (UnObjet: VObjet; Oui: Boolean);
   function    SelectionUnique:    VObjet;
   procedure   Usage               (UnUsage: VUsage);
   function    Valeur :            Integer;
   procedure   VerrouilleObjet     (UnObjet: VObjet);
   procedure   VideCardinal;
   procedure   VideContenu;

   procedure   lButtonDown         (var Msg: TMessage); virtual;
   procedure   lButtonDblClk       (var Msg: TMessage); virtual;
   procedure   lButtonUp           (var Msg: TMessage); virtual;
   procedure   lButtonShift        (var Msg: TMessage); virtual;
   procedure   MouseMove           (var Msg: TMessage); virtual;
   procedure   rButtonDown         (var Msg: TMessage); virtual;
   END;

{-------------------------------------------------------------------------}
        IMPLEMENTATION
{-------------------------------------------------------------------------}

{-------------------------------------------------------------------------
        TBIcones
-------------------------------------------------------------------------}
```

```
   CONSTRUCTOR TBIcones.Init
               (UnParent: PWindow; UnTitre: PChar; AX, AY, NbX, NbY : INTEGER);
   VAR BX, BY : INTEGER;
       Rang   : VObjet;
   BEGIN
      IF (NbX < 1) OR (NbX > 19) THEN NbX := 10;
      IF (NbY < 1) OR (NbY > 19) THEN NbY := 10;
      IF NbX * NbY > MaxIcones   THEN NbY := MaxIcones DIV NbX;
      TBoiteCases.Init (UnParent, UnTitre, TRUE,
         AX, AY, NbX, NbY, LargeurIcone + EntreIcones, HauteurIcone + EntreIcones);

      Retrait.X1 := 1;
      Retrait.Y1 := 1;
      Retrait.X2 := 0;
      Retrait.Y2 := 0;
      Inter.X    := EntreIcones;
      Inter.Y    := EntreIcones;
      EncreTitre := cr_fgTitreBIcones;
      FondTitre  := cr_bgTitreBIcones;
      FondBoite  := cr_bgFondBIcones;
   +--IF cr_bgFondBIcones <> cr_bgFondDefaut THEN BEGIN
   |     CreeBrosseSolide (BrosseFond, cr_bgFondBIcones);
   |     CreePlumeSolide (Effaceur, cr_bgFondBIcones);
   +--END;
      IF cr_bgTitreBIcones <> cr_bgTitreDefaut
      THEN CreeBrosseSolide (BrosseTitre, FondTitre);
      CreePlumeSolide (PlumeCase, Noir);
      GetMem (Contenu, Word(NbX * NbY * SizeOf(TCaseBIcones)));
      GetMem (AncContenu, Word(NbX * NbY * SizeOf(TCaseBIcones)));
      MetAZero (AncContenu^[0], NbX * NbY * SizeOf(TCaseBIcones));

      Orientation := af_Lignes;
      MetAZero (LigneLimite, SizeOf(LigneLimite));
      PlumeLimite := CreatePen (ps_Solid, 1, Rouge);
      OKLimite    := TRUE;
      MonoSelect  := TRUE;
      OKSelect    := [Bonbon..Armoire];
      OKDeselect  := [Bonbon..Armoire];
      OKOuvrir    := [Sachet..Armoire];
      OKFermer    := [Bonbon..Caisse];
      Base        := 10;
      SousTitre   := StrNew('Base: dix');
      Usage (ub_Addition);
      VideCardinal;
      Active := TRUE;
   END;
   {-------------------------------------------------------------------------}
   DESTRUCTOR TBIcones.Done;
+--BEGIN
|     FreeMem (Contenu, NbCases.X * NbCases.Y * SizeOf (TCaseBIcones));
|     FreeMem (AncContenu, NbCases.X * NbCases.Y * SizeOf (TCaseBIcones));
|     DeleteObject (PlumeLimite);
|     TBoiteCases.Done;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION  TBIcones.AffichageOK (N: TReprNombre): VAffichage;
   VAR Cases    : Byte;
       Lignes   : Byte;
       Colonnes : Byte;
       Rang     : VObjet;
+--BEGIN
|     Cases := 0;
|     FOR Rang := Bonbon TO ObjetMax DO
|        Cases := Cases + N[Rang];
|     IF Cases > TotalCases
|     THEN AffichageOK := af_Impossible
|  +--ELSE CASE Orientation OF
|  |    +--af_Lignes : BEGIN
|  |    |     Lignes := 0;
|  |    |     FOR Rang := Bonbon TO ObjetMax DO
|  |    |        Lignes := Lignes + ((N[Rang] + NbCases.X - 1) DIV NbCases.X);
|  |    |     IF Lignes <= NbCases.Y
|  |    |     THEN AffichageOK := af_Lignes
|  |    |     ELSE AffichageOK := af_ContLig;
|  |    +-----END;
```

```
|  |     +--af_Colonnes, af_Cordes : BEGIN
|  |     |      Colonnes := 0;
|  |     |      FOR Rang := Bonbon TO ObjetMax DO
|  |     |          Colonnes := Colonnes + ((N[Rang] + NbCases.Y - 1) DIV NbCases.Y);
|  |     |      IF Colonnes <= NbCases.X
|  |     |      THEN AffichageOK := Orientation
|  |     |      ELSE IF Orientation = af_Colonnes
|  |     |           THEN AffichageOK := af_ContCol
|  |     |           ELSE AffichageOK := af_ContCord
|  |     +-----END;
|  +-------END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.AfficheCase (DC: HDC; NoCase: INTEGER);
   VAR IdIcone : HIcon;
       Rect : TRect;
+--BEGIN
|  IF Peinte AND NOT Egalite (Contenu^[NoCase], AncContenu^[NoCase], SizeOf(TCaseBIcones))
|+-THEN BEGIN
||       CoordCase (NoCase, Rect);
||    +--WITH Rect DO BEGIN
||    |      Inc (Left, Inter.X DIV 2);
||    |      Inc (Top, Inter.Y DIV 2);
||    |      IF AncContenu^[NoCase].Objet <> Rien
||    |      THEN Rectangle (DC, Left, Top, Right, Bottom);
||    |  +--WITH Contenu^[NoCase] DO BEGIN
||    |  |  +--IF Objet <> Rien THEN BEGIN
||    |  |  |      IdIcone := StockIcone [Objet,Genre];
||    |  |  |      DrawIcon (DC, Left, Top, IdIcone);
||    |  |  +--END;
||    |  |     TBoiteCases.SelectionneCase (DC, NoCase, Select);
||    |  +--END;
||    +--END;
|+------END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.AfficheContenu (DC: HDC; Info: TPaintStruct);
   VAR NoCase : INTEGER;
       Rect   : TRect;
+--BEGIN
|+-IF Peinte THEN BEGIN
||    SelectObject (DC, Effaceur);
||    SelectObject (DC, BrosseFond);
||    FOR NoCase := 0 TO (NbCases.X * NbCases.Y - 1) DO
||        AfficheCase (DC, NoCase);
||    MetAZero (AncContenu^, NbCases.X * NbCases.Y * SizeOf(TCaseBIcones));
|| +--IF OKLimite THEN BEGIN
|| |  +--WITH LigneLimite DO BEGIN
|| |  |     SelectObject (DC, Effaceur);
|| |  |     MoveTo (DC, Left, Top);
|| |  |     LineTo (DC, Right, Bottom);
|| |  |+-CASE AffichageOK (Card[Total]) OF
|| |  ||+af_Lignes, af_ContLig : IF NbCases.X >= Base THEN BEGIN
|| |  |||    Top := Y1 + Bord.Y1 + 1;
|| |  |||    Bottom := Y2 - Bord.Y2 - 1;
|| |  |||    Left := X1 + Bord.X1 + (Base-1)*DimCase.X;
|| |  |||    Right := Left;
|| |  ||+---END;
|| |  ||+af_Colonnes, af_ContCol : IF NbCases.Y >= Base THEN BEGIN
|| |  |||    Top := Y2 - Bord.Y2 - (Base-1)*DimCase.Y - 1;
|| |  |||    Bottom := Top;
|| |  |||    Left := X1 + Bord.X1 + 1;
|| |  |||    Right := X2 - Bord.X2 - 1;
|| |  ||+---END;
|| |  ||+af_Cordes, af_ContCord : IF NbCases.Y >= Base THEN BEGIN
|| |  |||    Top := Y1 + Bord.Y1 + (Base-1)*DimCase.Y;
|| |  |||    Bottom := Top;
|| |  |||    Left := X1 + Bord.X1 + 1;
|| |  |||    Right := X2 - Bord.X2 - 1;
|| |  ||+---END;
|| |  |+-END;
|| |  |     SelectObject (DC, PlumeLimite);
|| |  |     MoveTo (DC, Left, Top);
|| |  |     LineTo (DC, Right, Bottom);
|| |  +--END;
|| +--END;
```

```
|+--END;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBIcones.CumulSelect (UnObjet: VObjet; Genres: TSetCategories): Word;
   VAR Result : Word;
       UnGenre  : VCategorie;
+--BEGIN
|     Result := 0;
|     IF Total IN Genres THEN Genres := [Resultat..Report];
|     FOR UnGenre := Resultat TO Report DO
|         IF UnGenre IN Genres THEN Inc (Result, NbSelect[UnObjet,UnGenre]);
|     CumulSelect := Result;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.DetailleSelection (UnObjet: VObjet; VAR Detail: TDetailSelect);
   VAR NoCase : INTEGER;
+--BEGIN
|     MetAZero (Detail, SizeOf (Detail));
|  +--FOR NoCase := 0 TO TotalCases - 1 DO BEGIN
|  |   +--WITH Contenu^[NoCase] DO BEGIN
|  |   |     +--IF (Objet = UnObjet) AND Select THEN BEGIN
|  |   |     |      Inc (Detail[Genre]);
|  |   |     |      Inc (Detail[Total]);
|  |   |     +--END;
|  |   +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION IndiceMax (C: TReprNombre): VObjet;
   VAR Ind    : VObjet;
       IndMax : VObjet;
       Max    : Byte;
+--BEGIN
|     IndMax := Bonbon;  Max := C [Bonbon];
|     FOR Ind := Sachet TO ObjetMax DO
|      +--IF C[Ind] > Max THEN BEGIN
|      |      IndMax := Ind;
|      |      Max := C[Ind]
|      +--END;
|     IndiceMax := IndMax;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBIcones.LargeurSelection : Byte;
   VAR Obj : VObjet;
       Resultat : Byte;
+--BEGIN
|     Resultat := 0;
|     FOR Obj := Bonbon TO Armoire DO
|         IF NbSelect[Obj,Total] > 0 THEN Inc (Resultat);
|     LargeurSelection := Resultat;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.MessageGerant
             (ID: Word; UnObjet: VObjet; UnGenre: VCategorie; Code: Word);
   VAR Msg : TMsgDidact;
+--BEGIN
|     WITH PFenetreVide(Fenetre)^ DO
|  +--IF Gerant <> NIL THEN BEGIN
|  |     MetAZero (Msg, SizeOf(Msg));
|  |  +--WITH Msg DO BEGIN
|  |  |     MOrigine := @Self;
|  |  |     MMessage := ID;
|  |  |     MAccepte := Code = rr_SansErreur;
|  |  |     MCode    := Code;
|  |  |     MObjet   := UnObjet;
|  |  |     MGenre   := UnGenre;
|  |  +--END;
|  |     Gerant^.RecoitMessage (Msg);
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBIcones.PrendCardinal (N: TCardinal; BaseNum: Byte): BOOLEAN;
   VAR NoCase  : INTEGER;
       Rang    : VObjet;
       Aff     : VAffichage;
       Nbr     : Byte;
```

```
      UnGenre : VCategorie;
      UnDC    : HDC;
      Msg     : TMessage;
      Chaine  : STRING[BaseMax+10];
      PChaine : ARRAY[1..BaseMax+10] OF CHAR;
+--BEGIN
|     FlButton := fl_Sans;
|
|     IF (BaseNum < 2) OR (BaseNum > BaseMax) THEN BaseNum := 10;
|  +--FOR Rang := Bonbon TO ObjetMax DO BEGIN
|  |      N[Total,Rang] := 0;
|  |      FOR UnGenre := Resultat TO Report DO
|  |          Inc (N[Total,Rang], N[UnGenre,Rang]);
|  +--END;
|
|     IF Egalite (N, Card, SizeOf (Card)) THEN Exit;
|
|     Aff  := AffichageOK (N[Total]);
|  +--CASE Aff OF
|  |+-af_Lignes, af_ContLig : BEGIN
|  ||    Move (Contenu^, AncContenu^, TotalCases * SizeOf(TCaseBIcones));
|  ||    VideContenu;  NoCase := 0;
|  || +--FOR Rang := ObjetMax DOWNTO Bonbon DO BEGIN
|  || |   +--FOR UnGenre := Resultat TO Report DO BEGIN
|  || |   |   +--FOR Nbr := 1 TO N[UnGenre,Rang] DO BEGIN
|  || |   |   |      Contenu^[NoCase].Objet := Rang;
|  || |   |   |      Contenu^[NoCase].Genre := UnGenre;
|  || |   |   |      Inc (NoCase);
|  || |   |   +--END;
|  || |   +--END;
|  || |   +--IF Aff = af_Lignes THEN BEGIN
|  || |   |     IF (NoCase MOD NbCases.X) > 0
|  || |   |     THEN NoCase := ((NoCase DIV NbCases.X) + 1) * NbCases.X;
|  || |   +--END;
|  || +--END;
|  |+----END;
|  |+-af_Colonnes, af_ContCol : BEGIN
|  ||    Move (Contenu^, AncContenu^, TotalCases * SizeOf(TCaseBIcones));
|  ||    VideContenu;  NoCase := (NbCases.Y-1)*NbCases.X;
|  || +--FOR Rang := ObjetMax DOWNTO Bonbon DO BEGIN
|  || |   +--FOR UnGenre := Resultat TO Report DO BEGIN
|  || |   |   +--FOR Nbr := 1 TO N[UnGenre,Rang] DO BEGIN
|  || |   |   |      Contenu^[NoCase].Objet := Rang;
|  || |   |   |      Contenu^[NoCase].Genre := UnGenre;
|  || |   |   |      NoCase := NoCase -{+} NbCases.X;
|  || |   |   |      IF NoCase < 0
|  || |   |   |      THEN NoCase := NoCase + NbCases.X*NbCases.Y + 1;
|  || |   |   +--END;
|  || |   +--END;
|  || |   +--IF Aff = af_Colonnes THEN BEGIN
|  || |   |     IF (NoCase DIV NbCases.X) <> (NbCases.Y - 1)
|  || |   |     THEN NoCase := (NoCase MOD NbCases.X) + (NbCases.Y-1)*NbCases.X + 1;
|  || |   +--END;
|  || +--END;
|  |+----END;
|  |+-af_Cordes, af_ContCord : BEGIN
|  ||    Move (Contenu^, AncContenu^, TotalCases * SizeOf(TCaseBIcones));
|  ||    VideContenu;  NoCase := 0;
|  || +--FOR Rang := ObjetMax DOWNTO Bonbon DO BEGIN
|  || |   +--FOR UnGenre := Resultat TO Report DO BEGIN
|  || |   |   +--FOR Nbr := 1 TO N[UnGenre,Rang] DO BEGIN
|  || |   |   |      Contenu^[NoCase].Objet := Rang;
|  || |   |   |      Contenu^[NoCase].Genre := UnGenre;
|  || |   |   |      NoCase := NoCase + NbCases.X;
|  || |   |   |      IF NoCase >= (NbCases.X * NbCases.Y)
|  || |   |   |      THEN NoCase := (NoCase MOD NbCases.X) + 1;
|  || |   |   +--END;
|  || |   +--END;
|  || |   +--IF Aff = af_Cordes THEN BEGIN
|  || |   |     IF (NoCase DIV NbCases.X) <> 0
|  || |   |     THEN NoCase := (NoCase MOD NbCases.X) + 1;
|  || |   +--END;
|  || +--END;
|  |+----END;
|  +--END;
|
```

```
|     IF Aff = af_Impossible
|     THEN PrendCardinal := FALSE
|  +--ELSE BEGIN
|  |       Active := TRUE;
|  |       Card := N;
|  |    +--IF BaseNum <> Base THEN BEGIN
|  |    |     Base := BaseNum;
|  |    |     Chaine := 'Base: ' + EnToutesLettres(Base);
|  |    |     StrPCopy (@PChaine, Chaine);
|  |    |     PrendSousTitre (@PChaine);
|  |    +--END;
|  |       PrendDC (UnDC);
|  |       AfficheContenu (UnDC, DessineTout);
|  |       LacheDC (UnDC);
|  |       PrendCardinal := TRUE;
|  +-------END;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBIcones.PrendValeur (Val: INTEGER; BaseNum: Byte): BOOLEAN;
   VAR N        : TReprNombre;
       Rang     : VObjet;
       Cases    : Byte;
       Cardinal : TCardinal;
       Signe    : ShortInt;
+--BEGIN
|     IF Val >= 0
|     THEN Signe := 1
|     ELSE BEGIN Signe := -1; Val := -Val END;
|     IF (BaseNum < 2) OR (BaseNum > BaseMax) THEN BaseNum := 10;
|
|     ConvertitNombre (Val, BaseNum, N, 4);
|
|     MetAZero (Cardinal, SizeOf(Cardinal));
|     IF Signe = 1 THEN Cardinal[NPos] := N ELSE Cardinal[NNeg] := N;
|     Cardinal[Total] := N;
|     PrendValeur := PrendCardinal (Cardinal, BaseNum);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.RegroupeSelection;
   TYPE VCas =
        (FarNiente,FermeRien,FermeNPos,FermeNNeg,FermeEPos,FermeENeg,Efface);
   VAR NoCase     : INTEGER;
       NouvCard   : TCardinal;
       Orig, Dest : VObjet;
       NTypes     : Byte;
       Icone      : VObjet;
       ObjSel     : VObjet;
       CardSel    : TDetailSelect;
       RestePlus  : Byte;
       ResteMoins : Byte;
       ResteReport: Byte;
       CasDeFigure: VCas;
+--BEGIN
|     IF NOT MonoSelect THEN Exit;
|     ObjSel := SelectionUnique;
|     CasDeFigure := FarNiente;
|  +--IF (LargeurSelection = 1) AND (ObjSel <> Rien) THEN BEGIN
|  |     Move (Card, NouvCard, SizeOf(TCardinal));
|  |     DetailleSelection (ObjSel, CardSel);
|  |
|  |     IF ((CardSel[EPos]+CardSel[ENeg]) = 0)
|  |  +--THEN BEGIN
|  |  |        IF ((CardSel[NPos]+CardSel[Report]) > 0)
|  |  |     +--THEN BEGIN
|  |  |     |        IF CardSel[NNeg] = (CardSel[NPos] + CardSel[Report])
|  |  |     |        THEN CasDeFigure := Efface
|  |  |     |     +--ELSE IF (ObjSel IN OKFermer) AND (CardSel[NNeg] = 0) THEN BEGIN
|  |  |     |     |           IF (CardSel[NPos] + CardSel[Report] = Base)
|  |  |     |     |           THEN CasDeFigure := FermeNPos
|  |  |     |     |           ELSE IF (CardSel[NPos] + CardSel[Report] > Base)
|  |  |     |     |                THEN CasDeFigure := FermeRien;
|  |  |     |     +-------END;
|  |  |     +-------END
|  |  |     +--ELSE BEGIN
|  |  |     |        IF CardSel[NNeg] = Base
|  |  |     |        THEN CasDeFigure := FermeNNeg
```

```
|  |  |     |       ELSE IF CardSel[NNeg] > Base
|  |  |     |            THEN CasDeFigure := FermeRien;
|  |  |     +-------END;
|  |  +-------END
|  |  +--ELSE BEGIN
|  |  |       IF CardSel[EPos] <> 0
|  |  |     +--THEN BEGIN
|  |  |     |     +--IF CardSel[Total] = CardSel[EPos] THEN BEGIN
|  |  |     |     |      IF (CardSel[EPos] = Base)
|  |  |     |     |      THEN CasDeFigure := FermeEPos
|  |  |     |     |      ELSE IF (CardSel[EPos] > Base)
|  |  |     |     |           THEN CasDeFigure := FermeRien;
|  |  |     |     +--END;
|  |  |     +-------END
|  |  |     +--ELSE BEGIN
|  |  |     |     +--IF CardSel[Total] = CardSel[ENeg] THEN BEGIN
|  |  |     |     |      IF CardSel[ENeg] = Base
|  |  |     |     |      THEN CasDeFigure := FermeENeg
|  |  |     |     |      ELSE IF CardSel[ENeg] > Base
|  |  |     |     |           THEN CasDeFigure := FermeRien;
|  |  |     |     +--END;
|  |  |     +-------END;
|  |  +-------END;
|  |
|  |     Orig := ObjSel;
|  |     IF Orig < Armoire THEN Dest := Succ (Orig);
|  |  +--CASE CasDeFigure OF
|  |  |+-FermeNPos : BEGIN
|  |  ||    Dec (NouvCard[NPos,Orig], CardSel[NPos]);
|  |  ||    Dec (NouvCard[Report,Orig], CardSel[Report]);
|  |  ||    Inc (NouvCard[SiFermer[NPos],Dest], 1);
|  |  ||    PrendCardinal (NouvCard, Base);
|  |  ||    MessageGerant (mg_Ferme, ObjSel, NPos, rr_SansErreur);
|  |  |+----END;
|  |  |+-FermeNNeg  : BEGIN
|  |  ||    Dec (NouvCard[NNeg,Orig], Base);
|  |  ||    Inc (NouvCard[SiFermer[NNeg],Dest], 1);
|  |  ||    PrendCardinal (NouvCard, Base);
|  |  ||    MessageGerant (mg_Ferme, ObjSel, NNeg, rr_SansErreur);
|  |  |+----END;
|  |  |+-FermeEPos : BEGIN
|  |  ||    Dec (NouvCard[EPos,Orig], Base);
|  |  ||    Inc (NouvCard[SiFermer[EPos],Dest], 1);
|  |  ||    PrendCardinal (NouvCard, Base);
|  |  ||    MessageGerant (mg_Ferme, ObjSel, EPos, rr_SansErreur);
|  |  |+----END;
|  |  |+-FermeENeg : BEGIN
|  |  ||    Dec (NouvCard[ENeg,Orig], Base);
|  |  ||    Inc (NouvCard[SiFermer[ENeg],Dest], 1);
|  |  ||    PrendCardinal (NouvCard, Base);
|  |  ||    MessageGerant (mg_Ferme, ObjSel, EPos, rr_SansErreur);
|  |  |+----END;
|  |  |+-FermeRien : BEGIN
|  |  ||    MessageGerant (mg_Ferme, ObjSel, Total, rr_SelectTrop);
|  |  |+----END;
|  |  |+-Efface    : BEGIN
|  |  ||    RestePlus  := Card[NPos,ObjSel] - CardSel[NPos];
|  |  ||    ResteMoins := Card[NNeg,ObjSel] - CardSel[NNeg];
|  |  ||    ResteReport:= Card[Report,ObjSel] - CardSel[Report];
|  |  ||    NouvCard[NPos,ObjSel] := RestePlus;
|  |  ||    NouvCard[NNeg,ObjSel] := ResteMoins;
|  |  ||    NouvCard[Report,ObjSel] := ResteReport;
|  |  ||    PrendCardinal (NouvCard, Base);
|  |  ||    MessageGerant (mg_Reduit, ObjSel, Total, rr_SansErreur);
|  |  |+----END;
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.SelectionneCase (DC: HDC; NoCase: INTEGER; Oui: BOOLEAN);
   VAR Detruit : BOOLEAN;
+--BEGIN
|     IF (NoCase < 0) OR (NoCase >= TotalCases) THEN Exit;
|     WITH Contenu^[NoCase] DO
|  +--IF Select XOR Oui THEN BEGIN
|  |     IF DC = 0
```

```
|  |  +--THEN BEGIN
|  |  |       PrendDC (DC);
|  |  |       Detruit := TRUE;
|  |  +-------END
|  |     ELSE Detruit := FALSE;
|  |     TBoiteCases.SelectionneCase (DC, NoCase, Oui);
|  |     IF Detruit THEN LacheDC (DC);
|  |     Select := Oui;
|  |     IF Oui
|  |  +--THEN BEGIN
|  |  |       FlButton := fl_Select;
|  |  |       Inc (NbSelect[Objet,Genre]);
|  |  |       Inc (NbSelect[Objet,Total]);
|  |  |       MessageGerant (mg_Select, Objet, Genre, rr_SansErreur);
|  |  +-------END
|  |  +--ELSE BEGIN
|  |  |       FlButton := fl_Deselect;
|  |  |       Dec (NbSelect[Objet,Genre]);
|  |  |       Dec (NbSelect[Objet,Total]);
|  |  |       MessageGerant (mg_Deselect, Objet, Genre, rr_SansErreur);
|  |  +-------END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.SelectionneObjet (UnObjet: VObjet; Oui: BOOLEAN);
   VAR NoCase : INTEGER;
       DC : HDC;
+--BEGIN
|     PrendDC (DC);
|     FOR NoCase := 0 TO TotalCases-1 DO
|         WITH Contenu^[NoCase] DO
|         +--IF Objet = UnObjet THEN BEGIN
|         |     TBoiteCases.SelectionneCase (DC, NoCase, Oui);
|         |     IF Oui
|         |  +--THEN BEGIN
|         |  |     +--IF NOT Select THEN BEGIN
|         |  |     |     Inc (NbSelect[Objet,Genre]);
|         |  |     |     Inc (NbSelect[Objet,Total]);
|         |  |     +--END;
|         |  +-------END
|         |  +--ELSE BEGIN
|         |  |     +--IF Select THEN BEGIN
|         |  |     |     Dec (NbSelect[Objet,Genre]);
|         |  |     |     Dec (NbSelect[Objet,Total]);
|         |  |     +--END;
|         |  +-------END;
|         |     Select := Oui;
|         +--END;
|     LacheDC (DC);
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBIcones.SelectionUnique : VObjet;
   VAR Obj : VObjet;
       Resultat : VObjet;
       Largeur  : Byte;
+--BEGIN
|     Largeur := 0;
|     Resultat := Rien;
|     FOR Obj := Bonbon TO Armoire DO
|      +--IF NbSelect[Obj,Total] > 0 THEN BEGIN
|      |     Inc (Largeur);
|      |     Resultat := Obj;
|      +--END;
|     IF Largeur = 1
|     THEN SelectionUnique := Resultat
|     ELSE SelectionUnique := Rien;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.Usage (UnUsage: VUsage);
   VAR Cat : VCategorie;
+--BEGIN
|  +--FOR Cat := Resultat TO Report DO BEGIN
|  |      SiOuvrir[Cat] := NPos;
|  |      SiFermer[Cat] := NPos;
|  +--END;
|  +--FOR Cat := NNEg TO ENeg DO BEGIN
```

```
|  |      SiOuvrir[Cat] := NNeg;
|  |      SiFermer[Cat] := NNeg;
|  +--END;
|
|  +--CASE UnUsage OF
|  |+-ub_Addition : BEGIN
|  ||    SiOuvrir[NPos]   := EPos;
|  ||    SiOuvrir[NNeg]   := ENeg;
|  ||    SiFermer[NPos]   := Report;
|  ||    SiFermer[ENeg]   := NNeg;
|  |+----END;
|  |+-ub_Soustraction : BEGIN
|  ||    SiOuvrir[NPos]   := Report;
|  ||    SiOuvrir[Report] := Report;
|  ||    SiOuvrir[NNeg]   := ENeg;
|  ||    SiFermer[EPos]   := Report;
|  ||    SiFermer[ENeg]   := NNeg;
|  |+----END;
|  |+-ub_Nombre : BEGIN
|  ||    SiOuvrir[NNeg]   := NNeg;
|  ||    SiFermer[NNeg]   := NNeg;
|  |+----END;
|  |+-ub_Statique : BEGIN
|  ||    OKOuvrir   := [];
|  ||    OKFermer   := [];
|  ||    OKSelect   := [];
|  ||    OKDeselect := [];
|  |+----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TBIcones.Valeur : INTEGER;
   VAR Nombre : INTEGER;
       Mult : INTEGER;
       Rang : VObjet;
+--BEGIN
|     Mult := 1;
|     Nombre := 0;
|  +--FOR Rang := Bonbon TO ObjetMax DO BEGIN
|  |      Nombre := Nombre +
|  |        ((Card[NPos,Rang] + Card[Resultat,Rang] - Card[NNeg,Rang]) * Mult);
|  |      Mult := Mult * INTEGER (Base);
|  +--END;
|     Valeur := Nombre;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.VerrouilleObjet (UnObjet: VObjet);
   VAR NouvCard : TCardinal;
+--BEGIN
|     FlButton := fl_Sans;
|     NouvCard := Card;
|     NouvCard[Resultat,UnObjet] :=
|        Card[NPos,UnObjet] + Card[Report,UnObjet];
|     NouvCard[NPos,UnObjet] := 0;
|     NouvCard[Report,UnObjet] := 0;
|     PrendCardinal (NouvCard, Base);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.VideCardinal;
   VAR NoCase : INTEGER;
       Rang   : VObjet;
       UnGenre  : VCategorie;
+--BEGIN
|     FOR Rang := Bonbon TO ObjetMax DO
|         FOR UnGenre := Resultat TO Total DO
|             Card[UnGenre,Rang] := 0;
|     VideContenu;
|     FOR Rang := Bonbon TO Armoire DO
|         FOR UnGenre := Resultat TO Total DO
|             NbSelect[Rang,UnGenre] := 0;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBIcones.VideContenu;
   VAR NoCase : INTEGER;
+--BEGIN
|     Active := FALSE;
```

```
|     MetAZero (Contenu^[0], TotalCases * SizeOf(TCaseBIcones));
|     MetAZero (NbSelect, SizeOf (NbSelect));
+--END;

   {-------------------------------------------------------------------------}

   PROCEDURE TBIcones.lButtonDown (VAR Msg: TMessage);
   { Selectionne / deselectionne une icone }
   VAR NoCase     : INTEGER;
       UnObjet    : VObjet;
       CaseSelect : BOOLEAN;
       CaseVerrou : BOOLEAN;
       UnGenre    : VCategorie;
       Indice     : INTEGER;
+--BEGIN
|     NoCase     := CasePointee (Msg);
|   +--IF (NoCase > -1) THEN BEGIN
|   |   +--WITH Contenu^[NoCase] DO BEGIN
|   |   |      UnObjet := Objet;
|   |   |      CaseSelect := Select;
|   |   |      UnGenre := Genre;
|   |   +--END;
|   |      IF UnObjet = Rien
|   |      THEN MessageGerant (mg_Select, UnObjet, UnGenre, rr_Objet)
|   |      ELSE IF NOT MonoSelect
|   |   +--THEN BEGIN
|   |   |        IF CaseSelect
|   |   |     +--THEN BEGIN
|   |   |     |        FlButton := fl_Deselect;
|   |   |     |        IF (UnObjet IN OKDeselect)
|   |   |     |        THEN SelectionneCase (0, NoCase, FALSE)
|   |   |     |        ELSE MessageGerant (mg_Deselect, UnObjet, UnGenre, rr_Objet);
|   |   |     +--------END
|   |   |     +--ELSE BEGIN
|   |   |     |        FlButton := fl_select;
|   |   |     |        IF (UnObjet IN OKSelect)
|   |   |     |     +--THEN BEGIN
|   |   |     |     |        IF (UnGenre <> Resultat)
|   |   |     |     |        THEN SelectionneCase (0, NoCase, TRUE)
|   |   |     |     |        ELSE MessageGerant (mg_Select, UnObjet, UnGenre, rr_Genre);
|   |   |     |     +--------END
|   |   |     |        ELSE MessageGerant (mg_Select, UnObjet, UnGenre, rr_Objet);
|   |   |     +--------END;
|   |   +--------END
|   |   +--ELSE BEGIN
|   |   |     +--CASE LargeurSelection OF
|   |   |     |  0 : IF (UnObjet IN OKSelect)
|   |   |     |    +--THEN BEGIN
|   |   |     |    |        IF (UnGenre <> Resultat)
|   |   |     |    |        THEN SelectionneCase (0, NoCase, TRUE)
|   |   |     |    |        ELSE MessageGerant (mg_Select, UnObjet, UnGenre, rr_Genre);
|   |   |     |    +--------END
|   |   |     |       ELSE MessageGerant (mg_Select, UnObjet, UnGenre, rr_Objet);
|   |   |     |  1 : IF UnObjet = SelectionUnique
|   |   |     |    +--THEN BEGIN
|   |   |     |    |        IF CaseSelect
|   |   |     |    |     +--THEN BEGIN
|   |   |     |    |     |        IF UnObjet IN OKDeselect
|   |   |     |    |     |        THEN SelectionneCase (0, NoCase, FALSE)
|   |   |     |    |     |        ELSE MessageGerant (mg_Deselect, UnObjet, UnGenre, rr_Objet);
|   |   |     |    |     +--------END
|   |   |     |    |     +--ELSE BEGIN
|   |   |     |    |     |        IF (UnObjet IN OKSelect)
|   |   |     |    |     |     +--THEN BEGIN
|   |   |     |    |     |     |        IF (UnGenre <> Resultat)
|   |   |     |    |     |     |     +--THEN BEGIN
|   |   |     |    |     |     |     |        IF ((CumulSelect (UnObjet,[NPos,NNeg,Report]) > 0) AND
|   |   |     |    |     |     |     |            (UnGenre IN [EPos,ENeg])) OR
|   |   |     |    |     |     |     |           ((CumulSelect (UnObjet,[EPos,ENeg]) > 0) AND
|   |   |     |    |     |     |     |            (UnGenre IN [NPos,NNeg,Report]))
|   |   |     |    |     |     |     |        THEN lButtonShift (Msg);
|   |   |     |    |     |     |     |        SelectionneCase (0, NoCase, TRUE)
|   |   |     |    |     |     |     +--------END
|   |   |     |    |     |     |        ELSE MessageGerant (mg_Select, UnObjet, UnGenre, rr_Genre);
|   |   |     |    |     |     +--------END
|   |   |     |    |     |        ELSE MessageGerant (mg_Select, UnObjet, UnGenre, rr_Objet);
```

```
|  |  |     |   |     +-------END;
|  |  |     |   +-------END
|  |  |     |   +--ELSE BEGIN
|  |  |     |   |       IF (UnObjet IN OKSelect) AND (UnGenre <> Resultat)
|  |  |     |   |     +--THEN BEGIN
|  |  |     |   |     |       lButtonShift (Msg);
|  |  |     |   |     |       If LargeurSelection = 0 then SelectionneCase (0, NoCase, TRUE);
|  |  |     |   |     +-------END
|  |  |     |   |       ELSE MessageGerant (mg_Select, UnObjet, UnGenre, rr_Objet);
|  |  |     |   +-------END;
|  |  |     +--END;
|  |  +-------END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------------}
   PROCEDURE TBIcones.lButtonDblClk (VAR Msg: TMessage);
+--BEGIN
|     rButtonDown (Msg);
+--END;
   {-------------------------------------------------------------------------------}
   PROCEDURE TBIcones.lButtonUp (VAR Msg: TMessage);
   { analyse la selection courante lorsque le bouton gauche est relache }
+--BEGIN
|     RegroupeSelection;
+--END;
   {-------------------------------------------------------------------------------}
   PROCEDURE TBIcones.lButtonShift (VAR Msg: TMessage);
   { Deselectionne tout }
   VAR NoCase : INTEGER;
       UnDC   : HDC;
+--BEGIN
|     PrendDC (UnDC);
|     FOR NoCase := 0 TO TotalCases - 1 DO
|  +--WITH Contenu^[NoCase] DO BEGIN
|  |      IF Select AND (Objet IN OKDeselect)
|  |   +--THEN BEGIN
|  |   |       SelectionneCase (UnDC, NoCase, FALSE);
|  |   |       Select := FALSE;
|  |   +-------END;
|  +--END;
|     LacheDC (UnDC);
+--END;
   {-------------------------------------------------------------------------------}
   PROCEDURE TBIcones.MouseMove (VAR Msg: TMessage);
   { deplacement de la souris lorsque le bouton gauche est enfonce }
   VAR NoCase     : INTEGER;
       UnObjet    : VObjet;
       UnGenre    : VCategorie;
       CaseSelect : BOOLEAN;
       CaseVerrou : BOOLEAN;
+--BEGIN
|     NoCase     := CasePointee (Msg);
|  +--IF NoCase > -1 THEN BEGIN
|  |  +--WITH Contenu^[NoCase] DO BEGIN
|  |  |     UnObjet := Objet;
|  |  |     CaseSelect := Select;
|  |  |     UnGenre := Genre;
|  |  +--END;
|  |     IF UnObjet <> Rien THEN
|  |  +--CASE FlButton OF
|  |  |  fl_select :
|  |  |  +--IF NOT CaseSelect THEN BEGIN
|  |  |  |     IF Monoselect
|  |  |  |  +--THEN BEGIN
|  |  |  |  |     +--IF (UnObjet = SelectionUnique) THEN BEGIN
|  |  |  |  |     |     IF ((UnGenre IN [NPos,NNeg,Report]) AND
|  |  |  |  |     |         (CumulSelect (UnObjet,[NPos,NNeg,Report]) > 0) OR
|  |  |  |  |     |        ((UnGenre IN [EPos,ENeg]) AND
|  |  |  |  |     |         (CumulSelect (UnObjet,[EPos,ENeg]) > 0)))
|  |  |  |  |     |     THEN SelectionneCase (0, NoCase, TRUE)
|  |  |  |  |     +--END;
|  |  |  |  +-------END
|  |  |  |  +--ELSE BEGIN
|  |  |  |  |     +--IF (UnObjet IN OKSelect) THEN BEGIN
|  |  |  |  |     |     IF (UnGenre <> Resultat) THEN SelectionneCase (0, NoCase, TRUE);
|  |  |  |  |     +--END;
```

```
|  |  |  |  +-------END;
|  |  |  +--END;
|  |  |  fl_Deselect :
|  |  |  +--IF CaseSelect THEN BEGIN
|  |  |  |     IF (UnObjet IN OKDeselect) THEN SelectionneCase (0, NoCase, FALSE);
|  |  |  +--END;
|  |  +--END;
|  +--END;
+--END;
   {--------------------------------------------------------------------------}
   PROCEDURE TBIcones.rButtonDown (VAR Msg: TMessage);
   { remplace une icone par d'autres de rang inferieur }
   VAR NoCase     : INTEGER;
       Orig, Dest : VObjet;
       NouvCard   : TCardinal;
       UnObjet    : VObjet;
       UnGenre    : VCategorie;
       Code       : Word;
+--BEGIN
|     NoCase := CasePointee (Msg);
|     IF (NoCase > -1) AND (FlButton = fl_Sans) THEN
|  +--WITH Contenu^[NoCase] DO BEGIN
|  |     UnObjet := Objet;
|  |     UnGenre := Genre;
|  |  +--CASE UnObjet OF
|  |  |  Rien   : BEGIN END;
|  |  |  Bonbon : MessageGerant (mg_Ouvre, UnObjet, UnGenre, rr_Objet);
|  |  |  Sachet..
|  |  |  Armoire:
|  |  |  +--IF (UnObjet IN OKOuvrir) THEN BEGIN
|  |  |  |     Move (Card, NouvCard, SizeOf(Card));
|  |  |  |     Orig := UnObjet;
|  |  |  |     Dest := Pred (UnObjet);
|  |  |  |     Dec (NouvCard[UnGenre,Orig], 1);
|  |  |  |  +--CASE UnGenre OF
|  |  |  |  |+-NPos : BEGIN
|  |  |  |  ||    Inc (NouvCard[SiOuvrir[NPos],Dest], Base);
|  |  |  |  ||    IF PrendCardinal (NouvCard, Base)
|  |  |  |  ||    THEN Code := rr_SansErreur ELSE Code := rr_PasDePlace;
|  |  |  |  ||    MessageGerant (mg_Ouvre, UnObjet, UnGenre, Code);
|  |  |  |  |+----END;
|  |  |  |  |+-Report : BEGIN
|  |  |  |  ||    Inc (NouvCard[SiOuvrir[Report],Dest], Base);
|  |  |  |  ||    IF PrendCardinal (NouvCard, Base)
|  |  |  |  ||    THEN Code := rr_SansErreur ELSE Code := rr_PasDePlace;
|  |  |  |  ||    MessageGerant (mg_Ouvre, UnObjet, UnGenre, Code);
|  |  |  |  |+----END;
|  |  |  |  |+-NNeg : BEGIN
|  |  |  |  ||    Inc (NouvCard[SiOuvrir[NNeg],Dest], Base);
|  |  |  |  ||    IF PrendCardinal (NouvCard, Base)
|  |  |  |  ||    THEN Code := rr_SansErreur ELSE Code := rr_PasDePlace;
|  |  |  |  ||    MessageGerant (mg_Ouvre, UnObjet, UnGenre, Code);
|  |  |  |  |+----END;
|  |  |  |  |  ELSE MessageGerant (mg_Ouvre, UnObjet, UnGenre, rr_Genre);
|  |  |  |  +--END
|  |  |  +--END ELSE MessageGerant (mg_Ouvre, UnObjet, UnGenre, rr_Objet);
|  |  |  ELSE MessageGerant (mg_Ouvre, UnObjet, UnGenre, rr_Objet);
|  |  +--END;
|  +--END;
+--END;

   {--------------------------------------------------------------------------
            INITIALISATION
   --------------------------------------------------------------------------}
   END.
```

OCalcAS.pas

```
UNIT OCalcAS;

{-------------------------------------------------------------------------}
          INTERFACE
{-------------------------------------------------------------------------}

USES Objects, OWindows, WinTypes, WinProcs, Strings,
     Globaux;

TYPE
{-------------------------------------------------------------------------}
          Objet TCalcAddSub
{-------------------------------------------------------------------------}

PCalcAddSub = ^TCalcAddSub;
TCalcAddSub = OBJECT (TObject)
   Operation      : Char;
   Base           : Byte;
   Nombre         : Array[1..2] of TReprNombre;
   ResultatOK     : TReprNombre;
   ResultatBrutOK : TReprNombre;
   ResultatVu     : TReprNombre;
   ReportVu       : TReprNombre;
   ReportOKDe     : Array[Bonbon..Armoire] of Boolean;
   ReportOKVers   : Array[Bonbon..Armoire] of Boolean;
   RangMax        : VObjet;
   constructor Init;
   procedure   EffectueCalcul;
   procedure   PrendValeurs (Val1: Integer; Op:Char; Val2: Integer; B: Byte);
   End;

{-------------------------------------------------------------------------}
          IMPLEMENTATION
{-------------------------------------------------------------------------}

CONSTRUCTOR TCalcAddSub.Init;
BEGIN
   Operation      := '+';
   Base           := 10;
   MetAZero (Nombre, SizeOf(Nombre));
   MetAZero (ResultatOK, SizeOf(ResultatOK));
   MetAZero (ResultatBrutOK, SizeOf(ResultatBrutOK));
   MetAZero (ResultatVu, SizeOf(ResultatVu));
   MetAZero (ReportVu, SizeOf(ReportVu));
   MetAZero (ReportOKDe, SizeOf(ReportOKDe));
   MetAZero (ReportOKVers, SizeOf(ReportOKVers));
   RangMax        := Rien;
END;
{-------------------------------------------------------------------------}
PROCEDURE TCalcAddSub.EffectueCalcul;
VAR N1, N2 : TReprNombre;
    Ind    : VObjet;
+--BEGIN
|     MetAZero (ResultatOK, SizeOf (ResultatOK));
|     MetAZero (ResultatBrutOK, SizeOf (ResultatBrutOK));
|     MetAZero (ReportOKDe, SizeOf (ReportOKDe));
|     MetAZero (ReportOKVers, SizeOf (ReportOKVers));
|  +--CASE Operation OF
|  |+-'+' : FOR Ind := Bonbon TO Armoire DO BEGIN
|  ||           ResultatBrutOK[Ind] := Nombre[1][Ind] + Nombre[2][Ind];
|  ||           Inc (ResultatOK[Ind], ResultatBrutOK[Ind]);
|  ||        +--IF (ResultatOK[Ind] DIV Base) = 1 THEN BEGIN
|  ||        |  +--IF Ind < Armoire THEN BEGIN
|  ||        |  |     ResultatOK[Succ(Ind)] := 1;
|  ||        |  |     ReportOKDe[Ind] := TRUE;
|  ||        |  |     ReportOKVers[Succ(Ind)] := TRUE;
|  ||        |  +--END;
|  ||        |     Dec (ResultatOK[Ind], Base);
|  ||        +--END;
|  |+-------END;
|  |+-'-' : FOR Ind := Bonbon TO Armoire DO BEGIN
|  ||           ResultatBrutOK[Ind] := Nombre[1][Ind] - Nombre[2][Ind];
|  ||           Inc (ResultatOK[Ind], ResultatBrutOK[Ind]);
```

```
|  ||        +--IF ResultatOK[Ind] < 0 THEN BEGIN
|  ||        |  +--IF Ind < Armoire THEN BEGIN
|  ||        |  |     ResultatOK[Succ(Ind)] := -1;
|  ||        |  |     ReportOKDe[Succ(Ind)] := TRUE;
|  ||        |  |     ReportOKVers[Ind] := TRUE;
|  ||        |  +--END;
|  ||        |     Inc (ResultatOK[Ind], Base);
|  ||        +--END;
|  |+-------END;
|  +--END;
|     RangMax := Bonbon;
|     FOR Ind := Bonbon TO Armoire DO
|         IF ResultatOK[Ind] <> 0 THEN RangMax := Ind;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TCalcAddSub.PrendValeurs
             (Val1: INTEGER; Op: CHAR; Val2: INTEGER; B: Byte);
   VAR Rang     : Vobjet;
       Commence : BOOLEAN;
+--BEGIN
|     IF (Base < 2) OR (Base > 16) THEN Base := 10 ELSE Base := B;
|     IF Op IN ['+','-'] THEN Operation := Op ELSE Operation := '+';
|     MetAZero (ReportVu, SizeOf(ReportVu));
|     MetAZero (ResultatVu, SizeOf(ResultatVu));
|     ConvertitNombre (Val1, Base, Nombre[1], 4);
|     ConvertitNombre (Val2, Base, Nombre[2], 4);
|     EffectueCalcul;
+--END;

   {-------------------------------------------------------------------------
        INITIALISATION
   -------------------------------------------------------------------------}
   END.
```

OBAddSub.pas

```
UNIT OBAddSub;

{-------------------------------------------------------------------------}
          INTERFACE
{-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, Strings,
     Globaux, OBCases, ODidact, OCalcAS;

TYPE

PBAddSub = ^TBAddSub;
TBAddSub = OBJECT (TBoiteCases)
   Contenu         : Array[0..24] of TMiniChaine;
   CaseSelect      : Set of 0..24;
   OKSelect        : Set of 0..24;
   OKDeselect      : Set of 0..24;
   EncreNormal     : TColorRef;
   EncreReport     : TColorRef;
   EncreResultat   : TColorRef;
   EncreErreur     : TColorRef;
   EncreSymbole    : TColorRef;
   BrosseBleue     : HBrush;
   BrosseVerte     : HBrush;
   BrosseNoire     : HBrush;
   Calcul          : TCalcAddSub;

   constructor Init              (UnParent: PWindow; UnTitre: PChar;
                                 AX, AY: Integer);
   destructor  Done;             virtual;
   procedure   AccepteRang       (Rang: VObjet; Oui: Boolean);
   procedure   AfficheCase       (DC: HDC; NoCase: Integer);
   procedure   AfficheContenu    (DC: HDC; Info: TPaintStruct); virtual;
   procedure   DetermineReports;
   procedure   PrendValeurs      (Val1:Integer; Op:Char; Val2:Integer; B:Byte);
   procedure   PrendReport       (Rang: VObjet; Rep: Byte);
   procedure   PrendResultat     (Rang: VObjet; Res: Byte);
   procedure   SelectionneCase   (DC: HDC; NoCase: Integer; Oui: Boolean); virtual;
   procedure   SelectionneRang   (Rang: VObjet; Oui: Boolean);
   procedure   VideContenu;

   procedure   lButtonDown       (var Msg: TMessage); virtual;
   procedure   lButtonUp         (var Msg: TMessage); virtual;
   END;

{-------------------------------------------------------------------------}
          IMPLEMENTATION
{-------------------------------------------------------------------------}

CONSTRUCTOR TBAddSub.Init
          (UnParent: PWindow; UnTitre: PChar; AX, AY: INTEGER);
VAR Dim : Byte;
BEGIN
   Dim := GetSystemMetrics (sm_cyCaption);
   TBoiteCases.Init (UnParent, UnTitre, TRUE, AX, AY, 5, 5, Dim, Dim);
   SousTitre := StrNew('Base: dix');
   Calcul.Init;
   VideContenu;
   CaseSelect    := [];
   OKSelect      := [];
   OKDeselect    := [];
   EncreTitre    := cr_fgTitreBAddSub;
   FondTitre     := cr_bgTitreBAddSub;
   FondBoite     := cr_bgFondBAddSub;
   IF cr_bgTitreBAddSub <> cr_bgTitreDefaut
   THEN CreeBrosseSolide (BrosseTitre, cr_bgTitreBAddSub);
   IF cr_bgFondBAddSub <> cr_bgFondDefaut
   THEN CreeBrosseSolide (BrosseFond, cr_bgFondBAddSub);
   EncreSymbole := Noir;
   EncreNormal  := Bleu;
   EncreReport  := VertFonce;
   EncreResultat:= Noir;
   EncreErreur  := Rouge;
```

```
      BrosseVerte  := CreateSolidBrush (VertFonce);
      BrosseBleue  := CreateSolidBrush (Bleu);
      BrosseNoire  := CreateSolidBrush (Noir);
   END;
   {-------------------------------------------------------------------------}
   DESTRUCTOR TBAddSub.Done;
+--BEGIN
|     DeleteObject (BrosseBleue);
|     DeleteObject (BrosseVerte);
|     DeleteObject (BrosseNoire);
|     inherited Done;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.AccepteRang (Rang: VObjet; Oui: BOOLEAN);
   VAR I, N: INTEGER;
+--BEGIN
|  +--IF Calcul.ReportOKVers[Rang] THEN BEGIN
|  |     IF Oui
|  |     THEN OKSelect := OKSelect + [Byte(5-Ord(Rang))]
|  |     ELSE OKSelect := OKSelect - [Byte(5-Ord(Rang))];
|  +--END;
|  +--FOR I := 1 TO 2 DO BEGIN
|  |      N := 5-Ord(Rang)+I*5;
|  |   +--IF Contenu[N] <> '' THEN BEGIN
|  |   |     IF Oui
|  |   |     THEN OKSelect := OKSelect + [Byte(N)]
|  |   |     ELSE OKSelect := OKSelect - [Byte(N)];
|  |   +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.AfficheCase (DC: HDC; NoCase: INTEGER);
   VAR Rect    : TRect;
       Chaine  : ARRAY[1..3] OF CHAR;
       Col, Li : Byte;
       Couleur : TColorRef;
+--BEGIN
|     IF NOT (NoCase IN [0..24]) THEN Exit;
|     CoordCase (NoCase, Rect);
|     Col := 1 + NoCase MOD NbCases.X;
|     Li  := 1 + NoCase DIV NbCases.X;
|
|     IF Contenu[NoCase] = '?'
|     THEN Couleur := EncreErreur
|  +--ELSE CASE Li OF
|  |  1 : Couleur := EncreReport;
|  |  2 : Couleur := EncreNormal;
|  |  3 : IF Col = 1 THEN Couleur := EncreSymbole ELSE Couleur := EncreNormal;
|  |  4 : Couleur := EncreSymbole;
|  |  5 : Couleur := EncreResultat;
|  +--END;
|
|     IF Byte(NoCase) IN CaseSelect
|  +--THEN BEGIN
|  |        SetBkColor (DC, Couleur);
|  |        SetTextColor (DC, FondBoite);
|  |    +--CASE NoCase OF
|  |    |  01..04 : SelectObject (DC, BrosseVerte);
|  |    |  06..09,
|  |    |  11..14 : SelectObject (DC, BrosseBleue);
|  |    |  10,
|  |    |  21..24 : SelectObject (DC, BrosseNoire);
|  |    |  ELSE SelectObject (DC, BrosseFond);
|  |    +--END;
|  +-------END
|  +--ELSE BEGIN
|  |        SetBkColor (DC, FondBoite);
|  |        SetTextColor (DC, Couleur);
|  |        SelectObject (DC, BrosseFond);
|  +-------END;
|
|     SelectObject (DC, SystemFont);
|     IF NoCase IN [0,5,10,15..20]
|     THEN SelectObject (DC, Effaceur)
|     ELSE SelectObject (DC, PlumeCase);
|     Rectangle (DC, Rect.Left, Rect.Top, Rect.Right, Rect.Bottom);
```

```
|     StrPCopy (@Chaine, Contenu[NoCase]);
|     DrawText (DC, @Chaine, Length(Contenu[NoCase]), Rect,
|               DT_Center OR DT_VCenter OR DT_SingleLine OR DT_NoClip);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.AfficheContenu (DC: HDC; Info: TPaintStruct);
   VAR Rect    : TRect;
       Gauche  : INTEGER;
       Droit   : INTEGER;
       Hauteur : INTEGER;
       NoCase  : INTEGER;
+--BEGIN
|     EffaceContenu (DC);
|     FOR NoCase := 0 TO 24 DO
|         AfficheCase(DC, NoCase);
|     SelectObject (DC, Plume);
|     CoordCase (15, Rect);
|     Gauche := Rect.Left;
|     Hauteur := (Rect.Top + Rect.Bottom) DIV 2;
|     CoordCase (19, Rect);
|     Droit := Rect.Right;
|     MoveTo (DC, Gauche, Hauteur);
|     LineTo (DC, Droit, Hauteur);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.DetermineReports;
+--BEGIN
|     Calcul.EffectueCalcul;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.PrendReport (Rang: VObjet; Rep: Byte);
   VAR UnDC : HDC;
       NoCase : INTEGER;
+--BEGIN
|+-WITH Calcul DO BEGIN
||    IF Rang > RangMax THEN Exit;
||    NoCase := 5 - Ord(Rang);
|| +--CASE Operation OF
|| |  '+' :
|| |     IF Rep = ch_Espace
|| |     THEN Contenu[NoCase] := ''
|| |     ELSE IF (Rep IN [0,1]) AND ReportOKVers[Rang]
|| |          THEN Contenu[NoCase] := Chiffre[Rep]
|| |          ELSE Contenu[NoCase] := '?';
|| |
|| |  '-' :
|| |     IF Rep = ch_Espace
|| |     THEN Contenu[NoCase] := ''
|| |     ELSE IF (Rep = Base) AND ReportOKVers[Rang]
|| |          THEN Contenu[NoCase] := '10'
|| |          ELSE Contenu[NoCase] := '?';
|| +--END;
|| +--IF Peinte THEN BEGIN
|| |     PrendDC (UnDC);
|| |     AfficheCase(UnDC,NoCase);
|| |     LacheDC (UnDC);
|| +--END;
|+-END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.PrendResultat (Rang: VObjet; Res: Byte);
   VAR NoCase : INTEGER;
       UnDC : HDC;
+--BEGIN
|+-WITH Calcul DO BEGIN
||    IF Rang > RangMax THEN Exit;
||    NoCase := 4 * NbCases.X + 5 - Ord(Rang);
||    IF Res = ResultatOK[Rang]
||    THEN Contenu[NoCase] := Chiffre[Res]
||    ELSE Contenu[NoCase] := '?';
|| +--IF Peinte THEN BEGIN
|| |     PrendDC (UnDC);
|| |     AfficheCase(UnDC,NoCase);
|| |     LacheDC (UnDC);
|| +--END;
|+-END;
```

```
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.PrendValeurs (Val1:INTEGER; Op:CHAR; Val2:INTEGER; B:Byte);
   VAR NoCase   : INTEGER;
       Col, Li  : INTEGER;
       Rang     : Vobjet;
       Commence : BOOLEAN;
       UnDC     : HDC;
       Chaine   : STRING[BaseMax+10];
       PChaine  : ARRAY[1..BaseMax+10] OF CHAR;
+--BEGIN
|+-WITH Calcul DO BEGIN
|| +--IF B <> Base THEN BEGIN
|| |     Base := B;
|| |     Chaine := 'Base: ' + EnToutesLettres(Base);
|| |     StrPCopy (@PChaine, Chaine);
|| |     PrendSousTitre (@PChaine);
|| +--END;
||    Calcul.PrendValeurs (Val1, Op, Val2, B);
||    VideContenu;
|| +--FOR Li := 2 TO 3 DO BEGIN
|| |      Commence := FALSE;
|| |      FOR Col := 2 TO 5 DO
|| |          IF Nombre[Li-1,ObjetDeRang[6-Col]] = 0
|| |       +--THEN BEGIN
|| |       |       NoCase := (Li-1)*NbCases.X + Col-1;
|| |       |       IF Commence THEN Contenu[NoCase] := '0' ELSE Contenu[NoCase] := '';
|| |       +-------END
|| |       +--ELSE BEGIN
|| |       |       Commence := TRUE;
|| |       |       NoCase := (Li-1)*NbCases.X + Col-1;
|| |       |       Contenu[NoCase] := Chiffre[ Nombre[Li-1,ObjetDeRang[6-Col]] ];
|| |       +-------END;
|| +--END;
||    IF Val1 = 0 THEN Contenu[09] := '0';
||    IF Val2 = 0 THEN Contenu[14] := '0';
||    DetermineReports;
|| +--IF Peinte THEN BEGIN
|| |     PrendDC (UnDC);
|| |     AfficheContenu (UnDC, DessineTout);
|| |     LacheDC (UnDC);
|| +--END;
|+-END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.SelectionneCase (DC: HDC; NoCase: INTEGER; Oui: BOOLEAN);
+--BEGIN
|     IF NOT (Byte(NoCase) IN [0..24]) THEN Exit;
|     IF Oui
|     THEN CaseSelect := CaseSelect + [Byte(NoCase)]
|     ELSE CaseSelect := CaseSelect - [Byte(NoCase)];
|     AfficheCase(DC, NoCase);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.SelectionneRang (Rang: VObjet; Oui: BOOLEAN);
   VAR I, N: INTEGER;
       UnDC: HDC;
+--BEGIN
|     PrendDC (UnDC);
|     IF Calcul.ReportOKVers[Rang]
|     THEN SelectionneCase (UnDC, (5-Ord(Rang)), Oui);
|  +--FOR I := 1 TO 2 DO BEGIN
|  |     N := 5-Ord(Rang)+I*5;
|  |     IF Contenu[N] <> '' THEN SelectionneCase (UnDC, N, Oui);
|  +--END;
|     LacheDC (UnDC);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.VideContenu;
   VAR NoCase: INTEGER;
+--BEGIN
|     CaseSelect := [];
|     FOR NoCase := 0 TO 24 DO
|         Contenu[NoCase] := '';
|     Contenu[10] := Calcul.Operation;
+--END;
```

```
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.lButtonDown (VAR Msg: TMessage);
   VAR MG : TMsgDidact;
       NoCase : INTEGER;
       Col, Li : Byte;
       UnDC : HDC;
+--BEGIN
|     FlButton := fl_Select;
|     NoCase := CasePointee (Msg);
|  +--IF NoCase <> -1 THEN BEGIN
|  |     MetAZero (MG, SizeOf(MG));
|  |  +--WITH MG DO BEGIN
|  |  |     MOrigine := @Self;
|  |  |     Col := NoCase MOD NbCases.X + 1;
|  |  |  +--CASE Col OF
|  |  |  |  1    : MObjet := Rien;
|  |  |  |  2..5 : MObjet := ObjetDeRang[6-Col];
|  |  |  +--END;
|  |  |     Li  := NoCase DIV NbCases.X + 1;
|  |  |  +--CASE Li OF
|  |  |  |  1 : MGenre := Report;
|  |  |  |  2 : MGenre := NPos;
|  |  |  |  3 : MGenre := NNeg;
|  |  |  |  4 : MGenre := Total;
|  |  |  |  5 : MGenre := Resultat;
|  |  |  +--END;
|  |  |     IF NoCase IN CaseSelect
|  |  |  +--THEN BEGIN
|  |  |  |        MMessage := mg_Deselect;
|  |  |  |        IF NoCase IN OKDeselect
|  |  |  |     +--THEN BEGIN
|  |  |  |     |        MCode := rr_SansErreur;
|  |  |  |     |        PrendDC (UnDC);
|  |  |  |     |        SelectionneCase (UnDC, NoCase, FALSE);
|  |  |  |     |        LacheDC (UnDC);
|  |  |  |     +-------END
|  |  |  |        ELSE MCode := rr_NoCase;
|  |  |  +-------END
|  |  |  +--ELSE BEGIN
|  |  |  |        MMessage := mg_Select;
|  |  |  |        IF NoCase IN OKSelect
|  |  |  |     +--THEN BEGIN
|  |  |  |     |        MCode := rr_SansErreur;
|  |  |  |     |        PrendDC (UnDC);
|  |  |  |     |        SelectionneCase (UnDC, NoCase, TRUE);
|  |  |  |     |        LacheDC (UnDC);
|  |  |  |     +-------END
|  |  |  |        ELSE MCode := rr_NoCase;
|  |  |  +-------END;
|  |  |     MAccepte := MCode = rr_SansErreur;
|  |  +--END;
|  |     WITH PFenetreVide(Fenetre)^ DO
|  |     IF Gerant <> NIL THEN Gerant^.RecoitMessage (MG);
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBAddSub.lButtonUp (VAR Msg: TMessage);
   VAR MG : TMsgDidact;
       NoCase : INTEGER;
       Col, Li : Byte;
+--BEGIN
|     WITH PFenetreVide(Fenetre)^ DO
|  +--IF Gerant <> NIL THEN BEGIN
|  |     NoCase := CasePointee (Msg);
|  |     Col := NoCase MOD NbCases.X + 1;
|  |     Li  := NoCase DIV NbCases.X + 1;
|  |  +--IF (NoCase <> -1) AND (FlButton = fl_Sans) THEN BEGIN
|  |  |     MetAZero (MG, SizeOf(MG));
|  |  |  +--WITH MG DO BEGIN
|  |  |  |     MOrigine := @Self;
|  |  |  |     MMessage := mg_Depose;
|  |  |  |     MAccepte := TRUE;
|  |  |  |     MCode    := rr_SansErreur;
|  |  |  |     MNoCase  := NoCase;
|  |  |  |  +--CASE Col OF
|  |  |  |  |  1    : MObjet := Rien;
```

```
|  |  |  |  |  2..5 : MObjet := ObjetDeRang[6-Col];
|  |  |  |  +--END;
|  |  |  |  +--CASE Li OF
|  |  |  |  |  1 : MGenre := Report;
|  |  |  |  |  2 : MGenre := NPos;
|  |  |  |  |  3 : MGenre := NNeg;
|  |  |  |  |  4 : MGenre := Total;
|  |  |  |  |  5 : MGenre := Resultat;
|  |  |  |  +--END;
|  |  |  +--END;
|  |  |     Gerant^.RecoitMessage (MG);
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------------
        INITIALISATION
   -------------------------------------------------------------------------------}
   END.
```

DSimAdd.rc

```
DLGSOMMECHIFFRES DIALOG 114, 57, 143, 126
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Somme des chiffres"
BEGIN
      CONTROL "", 106, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_GROUP | WS_TABSTOP,
99, 69, 21, 12
      CONTROL "Quelle est la somme des chiffres ci-dessous ?", -1, "STATIC", SS_CENTER | WS_CHILD |
WS_VISIBLE, 16, 8, 112, 21
      CONTROL "", -1, "STATIC", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE | WS_GROUP, 5, 31, 133, 55
      CONTROL "OK", 1, "BUTTON", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP,
53, 94, 37, 25
      CONTROL "x", 101, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 35, 39, 16, 8
      CONTROL "x", 102, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 35, 55, 16, 8
      CONTROL "x", 103, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 35, 71, 16, 8
      CONTROL "=", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 70, 71, 5, 8
      CONTROL "+", 104, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 24, 55, 5, 8
      CONTROL "+", 105, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 24, 71, 5, 8
END
```

DSimAdd.pas

```
   UNIT DSimAdd;

   {-------------------------------------------------------------------------}
           INTERFACE
   {-------------------------------------------------------------------------}

   {$R DSimAdd.RC}

   USES OWindows, WinTypes, WinProcs, WinDos, Strings, ODialogs, Globaux;

+--FUNCTION DialogueSommeChiffres (Fenetre: PWindow;
|          Digits: T3Chiffres; Op: CHAR; VAR NbEssais: Byte): BOOLEAN;
|
|  {-------------------------------------------------------------------------}
|          IMPLEMENTATION
|  {-------------------------------------------------------------------------}
|
|  TYPE
|
|  TBufferSomme = RECORD
|     C1,C2,C3 : ARRAY[0..2] OF CHAR;
|     Op1, Op2 : ARRAY[0..1] OF CHAR;
|     ESomme   : ARRAY[0..2] OF CHAR;
|     END;
|  (
|  PDlgSommeChiffres = ^TDlgSommeChiffres;
|  TDlgSommeChiffres = OBJECT (TDialog)
|     NbEssais: Byte;
|     SommeOK : Byte;
|     Buffer  : TBufferSomme;
|     constructor Init (AParent: PWindow; AName: PChar;
|                 Digits: T3Chiffres; ABuffer: TBufferSomme);
|     function    CanClose: Boolean; virtual;
|     procedure   OK (var Msg: TMessage); virtual id_First + idOK;
|     END;
|  }
|  {-------------------------------------------------------------------------
|       TDlgSommeChiffres
|  -------------------------------------------------------------------------}
|  CONSTRUCTOR TDlgSommeChiffres.Init
|              (AParent: PWindow; AName: PChar;
|              Digits: T3Chiffres; ABuffer: TBufferSomme);
|  CONST ID_Chiffre1 = 101;
|        ID_Chiffre2 = 102;
|        ID_Chiffre3 = 103;
|        ID_Op1      = 104;
|        ID_OP2      = 105;
|        ID_Somme    = 106;
|  VAR B1, B2, B3, O1, O2 : PStatic;
```

```
|      BSomme : PEdit;
+--BEGIN
|      TDialog.Init (AParent, AName);
|      NbEssais := 0;
|      SommeOK := Digits[1] + Digits[2] + Digits[3];
|      B1 := New (PStatic, InitResource(@Self, ID_Chiffre1, 3));
|      B2 := New (PStatic, InitResource(@Self, ID_Chiffre2, 3));
|      B3 := New (PStatic, InitResource(@Self, ID_Chiffre3, 3));
|      O1 := New (PStatic, InitResource(@Self, ID_Op1, 2));
|      O2 := New (PStatic, InitResource(@Self, ID_Op2, 2));
|      BSomme := New (PEdit, InitResource(@Self, ID_Somme, 3));
|      BSomme^.Validator := New (PRangeValidatorF, Init (0, 99));
|      Buffer := ABuffer;
|      TransferBuffer := @Buffer;
+--END;
   {-------------------------------------------------------------------------}
+--FUNCTION TDlgSommeChiffres.CanClose: BOOLEAN;
|  VAR Dummy     : INTEGER;
|      Valeur    : REAL;
|      Erreur    : Byte;
|      SValeur   : STRING[4];
|      Somme     : Byte;
|      Message   : TCommentaire;
+--BEGIN
|      TransferData (tf_GetData);
|      Inc (NbEssais);
|      SValeur := StrPas (Buffer.ESomme);
|      Val (SValeur, Valeur, Dummy);
|      Somme := Trunc (Valeur);
|      IF Somme = SommeOK
|      THEN CanClose := TRUE
|   +--ELSE BEGIN
|   |        IF (SommeOK = 0) AND (Somme > 0) THEN Erreur := 1
|   |        ELSE IF (SommeOK > 0) AND (Somme = 0) THEN Erreur := 2
|   |        ELSE IF (Somme > 19) THEN Erreur := 3
|   |        ELSE IF Abs(SommeOK - Somme) > 1 THEN Erreur := 4
|   |        ELSE Erreur := 5;
|   |        IF NbEssais >= 3
|   |        THEN StrPCopy (@Message, 'Le résultat correct est '+EnToutesLettres(SommeOK)
|   |             +'. Revois les tables d''addition.')
|   |     +--ELSE CASE Erreur OF
|   |     |        1 : StrPCopy (@Message,
|   |     |            'Les chiffres à additionner sont des 0.'+
|   |     |            ' Le résultat ne peut être que 0 !');
|   |     |        2 : StrPCopy (@Message,
|   |     |            'Au moins un des chiffres à additionner n''est pas 0.'+
|   |     |            ' Donc, le résultat n''est pas nul !');
|   |     |        3 : StrPCopy (@Message,
|   |     |            'Tu arrives à un résultat trop grand !');
|   |     |        4 : StrPCopy (@Message,
|   |     |            'Ce n''est pas ça, mais tu peux y arriver.');
|   |     |        5 : StrPCopy (@Message,
|   |     |            'C''est presque juste. Tu te trompes d''une unité.');
|   |     +-------END;
|   |        MessageBox (HWindow,@Message,'Erreur', mb_OK OR mb_IconInformation);
|   |        CanClose := NbEssais > 3
|   +-------END;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE TDlgSommeChiffres.OK (VAR Msg: TMessage);
+--BEGIN
|      TDialog.OK (Msg);
+--END;

   {-------------------------------------------------------------------------
      Gestion du buffer et des paramètres par la procédure appelante
   -------------------------------------------------------------------------}
+--FUNCTION DialogueSommeChiffres(Fenetre: PWindow;
|            Digits: T3Chiffres; Op: CHAR; VAR NbEssais: Byte): BOOLEAN;
|  VAR Dlg       : PDlgSommeChiffres;
|      OldBuffer : TBufferSomme;
|      Return    : INTEGER;
+--BEGIN
|      NbEssais := 0;
|      MetAZero (OldBuffer, SizeOf(OldBuffer));
|   +--WITH OldBuffer DO BEGIN
```

```
|  |     StrPCopy (C1, Chiffre[Digits[1]]);
|  |     StrPCopy (C2, Chiffre[Digits[2]]);
|  |     StrPCopy (C3, Chiffre[Abs(Digits[3])]);
|  |     Op1[0] := '+';
|  |     IF Op IN ['+','-'] THEN Op2[0] := Op ELSE Op2[0] := '+';
|  +--END;
|     Dlg := New (PDlgSommeChiffres,
|            Init (Fenetre, 'DlgSommeChiffres', Digits, OldBuffer));
|  +--REPEAT
|  |     Return := Dlg^.Execute;
|  |     Inc (NbEssais, Dlg^.NbEssais);
|  +--UNTIL (Return = IdOK);
|     DialogueSommeChiffres := TRUE;
|     Dispose (Dlg, Done);
+--END;
   {-------------------------------------------------------------------------}
   END.
```

OBSimAdd.pas

```
UNIT OBSimAdd;

{-------------------------------------------------------------------------}
         INTERFACE
{-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, Strings,
     Globaux, OBCaptur, ODidact, DSimAdd;

TYPE
{-------------------------------------------------------------------------
     Objet TBSimpleAdd
-------------------------------------------------------------------------}

PBSimpleAdd = ^TBSimpleAdd;
TBSimpleAdd = OBJECT (TBoiteCapture)
   Contenu     : Array[0..8] of TMiniChaine;
   OKValeurs   : Set of 1..3;
   Valeur      : T3Chiffres;
   Somme       : Integer;
   SommeDonnee : Boolean;
   Operation   : Char;
   QuiCalcule  : VChoix;
   constructor Init (UnParent: PWindow; UnTitre: PChar; AX, AY : Integer);
   procedure   AfficheCase     (DC: HDC; NoCase: Integer);
   procedure   AfficheContenu  (DC: HDC; UnPaint: TPaintStruct); virtual;
   procedure   CalculeSomme;
   procedure   MessageGerant   (ID: Word; NoCase: Integer; Code: Word); virtual;
   procedure   AjouteValeur    (Cat: VCategorie; UneValeur: Integer);
   procedure   VideContenu;    virtual;
   End;

{-------------------------------------------------------------------------}
         IMPLEMENTATION
{-------------------------------------------------------------------------}

CONST
CouleurCase : ARRAY[0..8] OF TColorRef
   = (Noir, Noir, Noir, Noir, Noir, Noir, Noir, Bleu, Bleu);

{-------------------------------------------------------------------------}

CONSTRUCTOR TBSimpleAdd.Init (UnParent: PWindow; UnTitre: PChar; AX, AY: INTEGER);
VAR Dim : Byte;
BEGIN
   Dim := GetSystemMetrics (sm_cyCaption);
   TBoiteCapture.Init (UnParent, UnTitre, FALSE, AX, AY, 9, 1, Dim, Dim);
   IF cr_bgTitreBSimAdd <> cr_bgTitreDefaut
   THEN CreeBrosseSolide (BrosseTitre, cr_bgTitreBSimAdd);
   IF cr_bgFondBSimAdd <> cr_bgFondDefaut
   THEN CreeBrosseSolide (BrosseFond, cr_bgFondBSimAdd);
   EncreTitre := cr_fgTitreBSimAdd;
   FondTitre  := cr_bgTitreBSimAdd;
   VideContenu;
```

```
      QuiCalcule := ch_Eleve;
   END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBSimpleAdd.AfficheCase (DC: HDC; NoCase: INTEGER);
   VAR Rect: TRect;
       PChaine : ARRAY[0..2] OF CHAR;
+--BEGIN
|     CoordCase (NoCase, Rect);
|     SelectObject (DC, SystemFont);
|     StrPCopy (@PChaine, Contenu[NoCase]);
|     SetTextColor (DC, CouleurCase[NoCase]);
|     SetBkColor (DC, FondBoite);
|     DrawText (DC, @PChaine, StrLen(@Pchaine), Rect,
|        DT_Center OR DT_VCenter OR DT_SingleLine);
|     IF NoCase IN OKCapture
|     THEN SelectionneCase (DC, NoCase, TRUE);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBSimpleAdd.AfficheContenu (DC: HDC; UnPaint: TPaintStruct);
   VAR NoCase : INTEGER;
+--BEGIN
|  +--IF Peinte THEN BEGIN
|  |     EffaceContenu (DC);
|  |     FOR NoCase := 0 TO 5 DO
|  |         AfficheCase (DC, NoCase);
|  |     IF SommeDonnee
|  |     THEN FOR NoCase := 6 TO 8 DO
|  |              AfficheCase (DC, NoCase);
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBSimpleAdd.CalculeSomme;
   VAR No : Byte;
       Chaine : TMiniChaine;
       UnDC : HDC;
       NbEssais: Byte;
+--BEGIN
|     IF (QuiCalcule = ch_Eleve) AND (BitsAl(OKValeurs, 1)>1)
|     THEN DialogueSommeChiffres (Fenetre, Valeur, Operation, NbEssais);
|     SommeDonnee := TRUE;
|     Somme := 0;
|     FOR No := 1 TO 3 DO
|         Inc (Somme, Valeur[No]);
|         Str (Somme:2, Chaine);
|         Contenu[6] := '=';
|         IF Chaine[1] = ' '
|      +--THEN BEGIN
|      |       Contenu[7] := '';
|      |       OKCapture := [8];
|      +-------END
|      +--ELSE BEGIN
|      |       Contenu[7] := Chaine[1];
|      |       OKCapture := [7,8];
|      +-------END;
|         Contenu[8] := Chaine[2];
|     PrendDC (UnDC);
|     AfficheContenu (UnDC, DessineTout);
|     LacheDC (UnDC);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBSimpleAdd.MessageGerant (ID: Word; NoCase: INTEGER; Code: Word);
   VAR MG : TMsgDidact;
+--BEGIN
|     WITH PFenetreVide(Fenetre)^ DO
|  +--IF Gerant <> NIL THEN BEGIN
|  |     MetAZero (MG, SizeOf(MG));
|  |  +--WITH MG DO BEGIN
|  |  |     MOrigine := @Self;
|  |  |     MMessage := ID;
|  |  |  +--IF ID = mg_Prend THEN BEGIN
|  |  |  |     IF (NoCase IN OKCapture) AND (Contenu[NoCase] <> '')
|  |  |  |  +--THEN BEGIN
|  |  |  |  |       MAccepte := Code = rr_SansErreur;
|  |  |  |  |       MCode    := Code;
|  |  |  |  +-------END
|  |  |  |  +--ELSE BEGIN
```

```
|  |  |  |  |       MAccepte := FALSE;
|  |  |  |  |       MCode    := rr_NoCase;
|  |  |  |  +-------END;
|  |  |  +--END;
|  |  |     MNoCase  := NoCase;
|  |  |     MObjet   := Rien;
|  |  |  +--CASE NoCase OF
|  |  |  |  7    : MGenre := Report;
|  |  |  |  8    : MGenre := Resultat;
|  |  |  |  ELSE   MGenre := Total;
|  |  |  +--END;
|  |  +--END;
|  |     Gerant^.RecoitMessage (MG);
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBSimpleAdd.AjouteValeur (Cat: VCategorie; UneValeur: INTEGER);
   VAR Chaine : TMiniChaine;
       NoCase : INTEGER;
       UnDC   : HDC;
+--BEGIN
|     IF OKValeurs = [1..3] THEN Exit;
|     IF (UneValeur < -9) OR (UneValeur > 10) THEN Exit;
|     IF NOT (Cat IN [NPos, NNeg, Report]) THEN Exit;
|     IF (UneValeur < 0) AND (Cat <> NNeg) THEN Exit;
|     IF (UneValeur > 0) AND (Cat = NNeg) THEN Exit;
|
|  +--CASE Cat OF
|  |+-Report : BEGIN
|  || +--CASE UneValeur OF
|  || |  0,1 : Contenu[1] := Chiffre[UneValeur];
|  || |+-10  : BEGIN
|  || ||    Contenu[0] := '1';
|  || ||    Contenu[1] := '0';
|  || |+----END;
|  || |  ELSE Exit
|  || +--END;
|  ||    Valeur[1] := UneValeur;
|  ||    IF 2 IN OKValeurs THEN Contenu[2] := '+';
|  ||    Include (OKValeurs, 1);
|  |+----END;
|  |+-NPos : BEGIN
|  ||    IF NOT (2 IN OKValeurs)
|  || +--THEN BEGIN
|  || |       Contenu[3] := Chiffre[UneValeur];
|  || |       Valeur[2] := UneValeur;
|  || |       IF 1 IN OKValeurs THEN Contenu[2] := '+';
|  || |       Include (OKValeurs, 2);
|  || +-------END
|  ||    ELSE IF NOT (3 IN OKValeurs)
|  ||      +--THEN BEGIN
|  ||      |       Contenu[5] := Chiffre[UneValeur];
|  ||      |       Valeur[3] := UneValeur;
|  ||      |       IF ([1,2] * OKValeurs) <> [] THEN Contenu[4] := '+';
|  ||      |       Include (OKValeurs, 3);
|  ||      |       Operation := '+';
|  ||      +-------END
|  ||         ELSE Exit
|  |+----END;
|  |+-NNeg : BEGIN
|  ||    Contenu[5] := Chiffre[-UneValeur];
|  ||    Contenu[4] := '-';
|  ||    Include (OKValeurs, 3);
|  ||    Operation := '+';
|  |+----END;
|  +--END;
|     SommeDonnee := FALSE;
|     PrendDC (UnDC);
|     AfficheContenu (UnDC, DessineTout);
|     LacheDC (UnDC);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBSimpleAdd.VideContenu;
   VAR NoCase : Byte;
       UnDC : HDC;
+--BEGIN
```

```
|     MetAZero (Contenu, SizeOf (Contenu));
|     MetAZero (Valeur, SizeOf (Valeur));
|     Operation  := '+';
|     OKValeurs  := [];
|     OKCapture  := [];
|     PrendDC (UnDC);
|     AfficheContenu (UnDC, DessineTout);
|     LacheDC (UnDC);
+--END;
   {-------------------------------------------------------------------------}

   {-------------------------------------------------------------------------
            INITIALISATION
   -------------------------------------------------------------------------}
   END.
```

DDidact.rc

```
DLGOPTIONSPRG DIALOG 13, 22, 208, 180
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Options générales"
BEGIN
      CONTROL "", -1, "STATIC", SS_GRAYRECT | WS_CHILD | WS_VISIBLE | WS_GROUP, 5, 5, 96, 50
      CONTROL "Combien de chiffres ?", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE, 7, 7, 92, 4
      CONTROL "&3 rangs maximum", 2101, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |
WS_GROUP | WS_TABSTOP, 12, 20, 84, 12
      CONTROL "&4 rangs maximum", 2102, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 12,
32, 84, 12
      CONTROL "", -1, "STATIC", SS_GRAYRECT | WS_CHILD | WS_VISIBLE | WS_GROUP, 107, 5, 96, 50
      CONTROL "Interroger sur les tables ?", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE, 109,
7, 92, 46
      CONTROL "oui, l'élève calcule", 2103, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |
WS_GROUP | WS_TABSTOP, 112, 20, 87, 12
      CONTROL "non, l'ordinateur calcule", 2104, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 112, 32, 87, 12
      CONTROL "", -1, "STATIC", SS_GRAYRECT | WS_CHILD | WS_VISIBLE | WS_GROUP, 5, 60, 96, 67
      CONTROL "Qui choisit les valeurs ?", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE, 7, 62,
92, 63
      CONTROL "L'&élève", 2105, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP |
WS_TABSTOP, 12, 75, 84, 12
      CONTROL "L'&instituteur", 2106, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 12, 87,
84, 12
      CONTROL "L'&ordinateur", 2107, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 12, 99,
84, 12
      CONTROL "", -1, "STATIC", SS_GRAYRECT | WS_CHILD | WS_VISIBLE | WS_GROUP, 107, 60, 96, 67
      CONTROL "Comment guider ?", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE, 109, 62, 92, 63
      CONTROL "&Pas d'explications", 2108, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |
WS_GROUP | WS_TABSTOP, 112, 75, 84, 12
      CONTROL "&Courtes explications", 2109, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE,
112, 87, 84, 12
      CONTROL "&Longues explications", 2110, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE,
112, 99, 84, 12
      CONTROL "", -1, "STATIC", SS_GRAYRECT | WS_CHILD | WS_VISIBLE | WS_GROUP, 5, 137, 198, 37
      CONTROL "OK", 1, "BUTTON", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP,
65, 145, 35, 20
      CONTROL "Annuler", 2, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 108, 145,
35, 20
END
```

DDidact.pas

```
UNIT DDidact;

{------------------------------------------------------------------------------}
        INTERFACE
{------------------------------------------------------------------------------}

{$R DDidact.RC}

USES OWindows, WinTypes, WinProcs, WinDos, Strings, ODialogs,
     Globaux;

+--FUNCTION DialogueOptionsPrg (Fenetre: PWindow; VAR Param: TParametresPrg) : BOOLEAN;
|
|  {------------------------------------------------------------------------------}
|          IMPLEMENTATION
|  {------------------------------------------------------------------------------}
|
|  CONST
|  {------------------------------------------------------------------------------
|       Items du dialogue Options|Générales
|  ------------------------------------------------------------------------------}
|  ID_3rangs        = 2101;
|  ID_4rangs        = 2102;
|  ID_EleveCalcule  = 2103;
|  ID_OrdiCalcule   = 2104;
|  ID_Eleve         = 2105;
|  ID_Professeur    = 2106;
|  ID_Ordinateur    = 2107;
```

```
|  ID_Absent        = 2108;
|  ID_Bref          = 2109;
|  ID_Detaille      = 2110;
|
|  TYPE
|  {-------------------------------------------------------------------------
|       Boîte de dialogue : état des différents boutons
|  -------------------------------------------------------------------------}
|  TBufferOptionsPrg = RECORD
|     b3rangs,
|     b4rangs,
|     bEleveCalcule,
|     bOrdiCalcule,
|     bEleve,
|     bProfesseur,
|     bOrdinateur,
|     bAbsent,
|     bBref,
|     bDetaille : Word;
|     END;
|  {
|  PDlgOptionsPrg = ^TDlgOptionsPrg;
|  TDlgOptionsPrg = OBJECT (TDialog)
|     Buffer : TBufferOptionsPrg;
|     constructor Init (AParent: PWindow; AName: PChar; ABuffer:TBufferOptionsPrg);
|     procedure   OK (var Msg: TMessage); virtual id_First + idOK;
|     END;
|  }
|  {-------------------------------------------------------------------------
|       TDglOptionsPrg
|  -------------------------------------------------------------------------}
|  CONSTRUCTOR TDlgOptionsPrg.Init
|              (AParent: PWindow; AName: PChar; ABuffer: TBufferOptionsPrg);
|  VAR b3rangs,
|      b4rangs,
|      bEleveCalcule,
|      bOrdiCalcule,
|      bEleve,
|      bProfesseur,
|      bOrdinateur,
|      bAbsent,
|      bBref,
|      bDetaille : PRadioButton;
|      BOK, BAnnuler: PButton;
+--BEGIN
|     TDialog.Init (AParent, AName);
|     B3rangs      := New (PRadioButton, InitResource(@Self, ID_3rangs));
|     B4rangs      := New (PRadioButton, InitResource(@Self, ID_4rangs));
|     BEleveCalcule:= New (PRadioButton, InitResource(@Self, ID_EleveCalcule));
|     BOrdiCalcule := New (PRadioButton, InitResource(@Self, ID_OrdiCalcule));
|     BEleve       := New (PRadioButton, InitResource(@Self, ID_Eleve));
|     BProfesseur  := New (PRadioButton, InitResource(@Self, ID_Professeur));
|     BOrdinateur  := New (PRadioButton, InitResource(@Self, ID_Ordinateur));
|     BAbsent      := New (PRadioButton, InitResource(@Self, ID_Absent));
|     BBref        := New (PRadioButton, InitResource(@Self, ID_Bref));
|     BDetaille    := New (PRadioButton, InitResource(@Self, ID_Detaille));
|     Buffer := ABuffer;
|     TransferBuffer := @Buffer;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE TDlgOptionsPrg.OK (VAR Msg: TMessage);
+--BEGIN
|     TDialog.OK (Msg);
|     TransferData (tf_GetData);
+--END;

   {-------------------------------------------------------------------------
      Gestion du buffer et des paramètres par la procédure appelante
   -------------------------------------------------------------------------}
+--PROCEDURE InitBuffer (VAR Buffer: TBufferOptionsPrg; Param: TParametresPrg);
+--BEGIN
|     MetAZero (Buffer, SizeOf(Buffer));
|  +--WITH Param, Buffer DO BEGIN
|  |  +--CASE NBRangs OF
|  |  |  3 : b3rangs := bf_Checked;
|  |  |  4 : b4rangs := bf_Checked;
```

```
|  |  +--END;
|  |  +--CASE Guide OF
|  |  |  gd_Absent   : bAbsent   := bf_Checked;
|  |  |  gd_Bref     : bBref     := bf_Checked;
|  |  |  gd_Detaille : bDetaille := bf_Checked;
|  |  +--END;
|  |  +--CASE Choix OF
|  |  |  ch_Eleve      : bEleve      := bf_Checked;
|  |  |  ch_Professeur : bProfesseur := bf_Checked;
|  |  |  ch_Ordinateur : bOrdinateur := bf_Checked;
|  |  +--END;
|  |  +--CASE Tables OF
|  |  |  ch_Eleve      : bEleveCalcule := bf_Checked;
|  |  |  ch_Ordinateur : bOrdiCalcule  := bf_Checked;
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE GetBufferValues (Buffer: TBufferOptionsPrg; VAR Param: TParametresPrg);
+--BEGIN
|  +--WITH Param, Buffer DO BEGIN
|  |     IF b3rangs      = bf_Checked THEN NBRangs := 3;
|  |     IF b4rangs      = bf_Checked THEN NBRangs := 4;
|  |     IF bAbsent      = bf_Checked THEN Guide := gd_Absent;
|  |     IF bBref        = bf_Checked THEN Guide := gd_Bref;
|  |     IF bDetaille    = bf_Checked THEN Guide := gd_Detaille;
|  |     IF bEleve       = bf_Checked THEN Choix := ch_Eleve;
|  |     IF bProfesseur  = bf_Checked THEN Choix := ch_Professeur;
|  |     IF bOrdinateur  = bf_Checked THEN Choix := ch_Ordinateur;
|  |     IF bEleveCalcule= bf_Checked THEN Tables:= ch_Eleve;
|  |     IF bOrdiCalcule = bf_Checked THEN Tables:= ch_Ordinateur;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
+--FUNCTION DialogueOptionsPrg (Fenetre: PWindow; VAR Param: TParametresPrg) : BOOLEAN;
|  VAR Dlg       : PDlgOptionsPrg;
|      OldBuffer : TBufferOptionsPrg;
+--BEGIN
|     InitBuffer (OldBuffer, Param);
|     Dlg := New (PDlgOptionsPrg, Init (Fenetre, 'DLGOPTIONSPRG', OldBuffer));
|     IF Dlg^.Execute = IdOK
|  +--THEN BEGIN
|  |       GetBufferValues (Dlg^.Buffer, Param);
|  |       IF NOT Egalite (Dlg^.Buffer, OldBuffer, SizeOf(OldBuffer))
|  |       THEN DialogueOptionsPrg := TRUE;
|  +-------END
|     ELSE DialogueOptionsPrg := FALSE;
|     Dispose (Dlg, Done);
+--END;
   {-------------------------------------------------------------------------}
   END.
```

OBouton.pas

```
   UNIT OBouton;

   {-------------------------------------------------------------------------}
        INTERFACE
   {-------------------------------------------------------------------------}

   USES WinProcs, WinTypes, OWindows, ODialogs,
        Globaux;

   TYPE

   {-------------------------------------------------------------------------
        TBOUTON
   -------------------------------------------------------------------------}
   PBouton = ^TBouton;
   TBouton = OBJECT(TButton)
      function     DonneTexte:      PChar;
      procedure    DevientDefaut;
      procedure    DevientNormal;
      procedure    Enfonce;
      function     EstActif:        Boolean;
      procedure    GetWindowClass   (var AWndClass: TWndClass); virtual;
      procedure    PrendTexte       (Texte : PChar);
      procedure    RecoitFocus      (OuiNon: Boolean);
      procedure    RendActif        (OuiNon: Boolean);
      End;

   {-------------------------------------------------------------------------}
        IMPLEMENTATION
   {-------------------------------------------------------------------------}

   USES ODidact;

   CONST
   TempsEnfonce = 500; (* en millièmes de seconde *)

   {-------------------------------------------------------------------------}

+--FUNCTION TBouton.DonneTexte: PChar;
+--BEGIN
|     DonneTexte := Attr.Title
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE TBouton.DevientDefaut;
+--BEGIN
|     SendMessage(HWindow,bm_SetStyle,bs_DefPushButton,1);
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE TBouton.DevientNormal;
+--BEGIN
|     SendMessage(HWindow,bm_SetStyle,bs_PushButton,1)
+--END;
   {-------------------------------------------------------------------------}
+--PROCEDURE TBouton.Enfonce;
+--BEGIN
|     SendMessage (Hwindow,bm_SetState,1,0);
|     IF SetTimer (HWindow,0,TempsEnfonce,NIL) <> 0
|  +--THEN BEGIN
|  |       WaitMessage;
|  |       KillTimer (HWindow,0);
|  +-------END
|  +--ELSE BEGIN{
|  |       MessageBox (HWindow, 'No Timers Left', 'Error', mb_Ok);
|  |       Halt(1);}
|  +-------END;
|     SendMessage (HWindow,bm_SetState,0,0)
+--END;
   {-------------------------------------------------------------------------}
+--FUNCTION TBouton.EstActif: BOOLEAN;
+--BEGIN
|    EstActif := IsWindowEnabled(HWindow)
+--END;
   {-------------------------------------------------------------------------}
```

```
+--PROCEDURE TBouton.GetWindowClass (VAR AWndClass: TWndClass);
+--BEGIN
|     TButton.GetWindowClass (AWndClass);
|  +--WITH AWndClass DO BEGIN
|  |     hbrBackGround := Color_BtnFace + 1;
|  +--END;
+--END;
   {--------------------------------------------------------------------------}
+--PROCEDURE TBouton.PrendTexte (Texte: PChar);
+--BEGIN
|     SetWindowText(HWindow, texte)
+--END;
   {--------------------------------------------------------------------------}
+--PROCEDURE TBouton.RecoitFocus (OuiNon: BOOLEAN);
+--BEGIN
|     IF OuiNon
|     THEN SetFocus(HWindow)
|     ELSE SendMessage(HWindow, wm_KillFocus, 0, 0);
+--END;
   {--------------------------------------------------------------------------}
+--PROCEDURE TBouton.RendActif (OuiNon: BOOLEAN);
+--BEGIN
|     EnableWindow (HWindow, OuiNon)
+--END;

   {--------------------------------------------------------------------------
        INITIALISATION
   --------------------------------------------------------------------------}
   END.
```

ODidact.pas

```
UNIT ODidact;

{-------------------------------------------------------------------------}
         INTERFACE
{-------------------------------------------------------------------------}

{$R MDidact.RC}

USES OWindows, WinTypes, WinProcs, WinDos, Strings, ODialogs,
     Globaux, OBoite, OBouton, DDidact;

CONST
{-------------------------------------------------------------------------
     Parametres
-------------------------------------------------------------------------}
MaxBoites        = 10;
MaxBoutons       = 05;

{-------------------------------------------------------------------------
     Items de menu et identificateurs
-------------------------------------------------------------------------}
cm_Choisir       = 101;
cm_Debut         = 102;
cm_Quitter       = 103;
cm_OptionsGen    = 201;
cm_OptionsEx     = 202;
cm_AideGen       = 301;
cm_AideEx        = 302;

id_Bouton01      = 1001;
id_Bouton02      = 1002;
id_Bouton03      = 1003;
id_Bouton04      = 1004;
id_Bouton05      = 1005;

TYPE
{-------------------------------------------------------------------------
     Pointeurs
-------------------------------------------------------------------------}
PDidactVide      = ^TDidactVide;
PGerantVide      = ^TGerantVide;
PFenetreVide     = ^TFenetreVide;

{-------------------------------------------------------------------------
    TDidactVide : application Windows qui utilise les objets didactiques
-------------------------------------------------------------------------}

TDidactVide = OBJECT (TApplication)
   Icone : HIcon;
   constructor Init (AName: PChar; AnIcon: PChar);
   procedure   InitMainWindow; virtual;
   END;

{-------------------------------------------------------------------------
    TGerantVide : objet générique permettant de garnir la fenêtre
    principale et d'assurer la coordination en les éléments qui y figurent
-------------------------------------------------------------------------}

TGerantVide = OBJECT
   Fenetre  : PFenetreVide;
   Stop     : Boolean;
   constructor Init;
   destructor  Done;              virtual;
   procedure   AideExercice;      virtual;
   procedure   Avertissement      (ID : Integer);
   procedure   ChangeParametres;  virtual;
   procedure   ClickBouton        (No: Byte); virtual;
   procedure   DoneExercice;      virtual;
   procedure   GarnitMainWindow;  virtual;
   procedure   InitExercice       (Param: Pointer); virtual;
   procedure   ModifieMenu;       virtual;
   procedure   RecoitMessage      (var Msg: TMsgDidact); virtual;
   procedure   VideMainWindow;    virtual;
```

```
   END;

{-------------------------------------------------------------------------
    TTableau... : contenu de la fenêtre principale
-------------------------------------------------------------------------}
TTableauBoites  = ARRAY[1..MaxBoites] OF PBoite;
TTableauBoutons = ARRAY[1..MaxBoutons] OF PBouton;

{-------------------------------------------------------------------------
    TFenetreVide : fenêtre (principale) du didacticiel
-------------------------------------------------------------------------}

TFenetreVide = OBJECT (TWindow)
   Gerant     : PGerantVide;
   FocusGauche: Byte;
   FocusDroit : Byte;
   Position   : LongInt;
   NBBoites   : Byte;
   Boite      : TTableauBoites;
   NBBoutons  : Byte;
   Bouton     : TTableauBoutons;
   Parametres : TParametresPrg;
   AideChargee: Boolean;

   constructor Init             (UnParent: PWindowsObject; UnTitre: PChar);
   destructor  Done;            virtual;

   procedure   AideGenerale     (var Msg: TMessage); virtual cm_First + cm_AideGen;
   procedure   AideExercice     (var Msg: TMessage); virtual cm_First + cm_AideEx;
   function    AjouteBoite      (Adresse : PBoite): Bool;
   function    AjouteBouton     (var Adresse : PBouton; Texte: PChar;
                                AX, AY, BX, BY: Integer): Bool;
   procedure   Message          (Titre, Texte: PChar); virtual;
   function    CanClose :       Boolean; virtual;
   procedure   DebutExercice    (var Msg: TMessage); virtual cm_First + cm_Debut;
   procedure   GetWindowClass   (var AWndClass: TWndClass); virtual;
   procedure   NomFAideGenerale (Fichier: PChar);
   procedure   OptionsGenerales (var Msg: TMessage); virtual cm_First + cm_OptionsGen;
   procedure   OptionsExercice  (var Msg: TMessage); virtual cm_First + cm_OptionsEx;
   procedure   Paint            (DC: HDC; var Info: TPaintStruct); virtual;
   procedure   Quitter          (var Msg: TMessage); virtual cm_First + cm_Quitter;
   procedure   SetUpWindow;     virtual;

   procedure   ClickBouton01    (var Msg: TMessage); virtual id_First + id_Bouton01;
   procedure   ClickBouton02    (var Msg: TMessage); virtual id_First + id_Bouton02;
   procedure   ClickBouton03    (var Msg: TMessage); virtual id_First + id_Bouton03;
   procedure   ClickBouton04    (var Msg: TMessage); virtual id_First + id_Bouton04;
   procedure   ClickBouton05    (var Msg: TMessage); virtual id_First + id_Bouton05;

   procedure   WMCancelMode     (var Msg: TMessage); virtual wm_First + wm_CancelMode;
   procedure   WMlButtonDblClk  (var Msg: TMessage); virtual wm_First + wm_lButtonDblClk;
   procedure   WMlButtonDown    (var Msg: TMessage); virtual wm_First + wm_lButtonDown;
   procedure   WMlButtonShift   (var Msg: TMessage); virtual;
   procedure   WMlButtonUp      (var Msg: TMessage); virtual wm_First + wm_lButtonUp;
   procedure   WMMouseMove      (var Msg: TMessage); virtual wm_First + wm_MouseMove;
   procedure   WMNClButtonUp    (var Msg: TMessage); virtual wm_First + wm_NClButtonUp;
   procedure   WMNCrButtonUp    (var Msg: TMessage); virtual wm_First + wm_NCrButtonUp;
   procedure   WMrButtonDown    (var Msg: TMessage); virtual wm_First + wm_rButtonDown;
   procedure   WMrButtonUp      (var Msg: TMessage); virtual wm_First + wm_rButtonUp;
   END;

{-------------------------------------------------------------------------}
        IMPLEMENTATION
{-------------------------------------------------------------------------}

{-------------------------------------------------------------------------
        TDidactVide
-------------------------------------------------------------------------}

CONSTRUCTOR TDidactVide.Init (AName: PChar; AnIcon: PChar);
BEGIN
   Icone := LoadIcon (HInstance, AnIcon);
   TApplication.Init(AName);
{   HAccTable := LoadAccelerators (HInstance, 'ACCELDIDACTIC');}
END;
{-------------------------------------------------------------------------}
```

```
   PROCEDURE TDidactVide.InitMainWindow;
+--BEGIN
|     MainWindow := New (PFenetreVide, Init (NIL, 'Didacticiel'));
+--END;

   {-------------------------------------------------------------------------
            TFenetreVide
   -------------------------------------------------------------------------}

   constructor TFenetreVide.Init (UnParent: PWindowsObject; UnTitre: PChar);
   VAR I : INTEGER;
+--BEGIN
|     TWindow.Init (UnParent, UnTitre);
|  +--WITH Attr DO BEGIN
|  |     Style := Style OR ws_Maximize;
|  |     Menu  := LoadMenu (HInstance, 'MenuDidactic');
|  |     X := 0;  Y := 0;  W := 600;  H := 400;
|  |     LoadAccelerators (HInstance, 'ACCELDIDACTIC');
|  +--END;
|
|     NBBoites := 0;
|     FocusGauche := 0;
|     FocusDroit  := 0;
|     Position    := 0;
|     FOR I:=1 TO MaxBoites DO Boite[I] := NIL;
|     NBBoutons := 0;
|     FOR I:=1 TO MaxBoutons DO Bouton[I] := NIL;
|     AideChargee := FALSE;
|     Gerant := NIL {New (PGerantVide, Init)};
|  +--WITH Parametres DO BEGIN
|  |     NBRangs := 4;
|  |     Guide   := gd_Detaille;
|  |     Choix   := ch_Ordinateur;
|  |     Tables  := ch_Eleve;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   destructor TFenetreVide.Done;
   VAR I : INTEGER;
       FAide : ARRAY[1..100] OF CHAR;
+--BEGIN
|     IF Gerant <> NIL THEN Dispose (Gerant, Done);
|     FOR I:=1 TO NBBoites DO
|      +--IF Boite[I] <> NIL THEN BEGIN
|      |     Dispose (Boite[I], Done);
|      |     Boite[I] := NIL;
|      +--END;
|     FOR I:=1 TO NBBoutons DO
|      +--IF Bouton[I] <> NIL THEN BEGIN
|      |     Dispose (Bouton[I], Done);
|      |     Bouton[I] := NIL;
|      +--END;
|  +--IF AideChargee THEN BEGIN
|  |     NomFAideGenerale (@FAide);
|  |     WinHelp (HWindow, @FAide, Help_Quit, 0);
|  +--END;
|     TWindow.Done;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.AideExercice (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Gerant^.AideExercice;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.AideGenerale (VAR Msg: TMessage);
+--BEGIN
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TFenetreVide.AjouteBoite (Adresse : PBoite): Bool;
+--BEGIN
|     IF NBBoites < MaxBoites
|  +--THEN BEGIN
|  |       Inc (NBBoites);
|  |       Boite [NBBoites] := Adresse;
|  |       AjouteBoite := TRUE;
|  +-------END
```

```
|     ELSE AjouteBoite := FALSE;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TFenetreVide.AjouteBouton
            (VAR Adresse : PBouton; Texte: PChar; AX, AY, BX, BY: INTEGER): Bool;
   VAR Id : Word;
+--BEGIN
|     IF NBBoutons < MaxBoutons
|  +--THEN BEGIN
|  |       Inc (NBBoutons);
|  |    +--CASE NBBoutons OF
|  |    |  1 : Id := Id_Bouton01;
|  |    |  2 : Id := Id_Bouton02;
|  |    |  3 : Id := Id_Bouton03;
|  |    |  4 : Id := Id_Bouton04;
|  |    |  5 : Id := Id_Bouton05;
|  |    +--END;
|  |       Bouton [NBBoutons] := New (PBouton,
|  |          Init(@Self, Id, Texte, AX, AY, BX-AX, BY-AY, FALSE));
|  |       Adresse := Bouton[NBBoutons];
|  |       AjouteBouton := TRUE;
|  +-------END
|  +--ELSE BEGIN
|  |       AjouteBouton := FALSE;
|  |       Adresse := NIL;
|  +-------END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.Message (Titre, Texte: PChar);
+--BEGIN
|     MessageBox (Self.HWindow, Texte, Titre, mb_OK OR mb_IconExclamation);
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TFenetreVide.CanClose;
   CONST Titre : ARRAY[1..15] OF CHAR = 'Confirmation';
   VAR   Texte : ARRAY[1..35] OF CHAR;
+--BEGIN
|     MetAZero(Texte, SizeOf(Texte));
|     IF @Self = Application^.MainWindow
|     THEN StrCat (@Texte, 'Veux-tu quitter le programme ?')
|     ELSE StrCat (@Texte, 'Veux-tu quitter l''exercice ?');
|     CanClose := TWindow.CanClose AND
|                  (MessageBox(HWindow,@Texte,@Titre,mb_YesNo OR mb_IconQuestion)
|                   = idYes)
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.DebutExercice (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Gerant^.InitExercice (NIL);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.GetWindowClass (VAR AWndClass: TWndClass);
+--BEGIN
|     TWindow.GetWindowClass(AWndClass);
|  +--WITH AWndClass DO BEGIN
|  |     hIcon := PDidactVide(Application)^.Icone;
|  |     Style := cs_SaveBits OR
|  |        cs_ByteAlignWindow OR cs_ByteAlignClient OR cs_DblClks;
|  |     hbrBackGround := CreateSolidBrush (cr_bgFondFenetre);
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.NomFAideGenerale (Fichier: PChar);
   VAR P, N, E : ARRAY[1..100] OF CHAR;
+--BEGIN
|     GetModuleFileName (HInstance, @P, 100);
|     FileSplit (@P, Fichier, @N, @E);
|     StrCat (Fichier, 'DIDACT.HLP')
+--END;

   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.OptionsGenerales (VAR Msg: TMessage);
   VAR Change : BOOLEAN;
+--BEGIN
|     IF DialogueOptionsPrg (@Self, Parametres) AND (Gerant <> NIL)
|     THEN Gerant^.ChangeParametres;
```

```
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.OptionsExercice (VAR Msg: TMessage);
+--BEGIN
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.Paint (DC: HDC; VAR Info: TPaintStruct);
   VAR I : INTEGER;
       Rect : TRect;
       Inter : TRect;
       PtInfo2 : TPaintStruct;
+--BEGIN
|  +--WITH Info DO BEGIN
|  |  +--IF NOT IsRectEmpty (rcPaint) THEN BEGIN
|  |  |  +--FOR I:=1 TO NBBoites DO BEGIN
|  |  |  |  |  +--IF Boite[I] <> NIL THEN BEGIN
|  |  |  |  |  |     Move (Boite[I]^.X1, Rect, SizeOf(Rect));
|  |  |  |  |  |  +--IF IntersectRect (Inter, rcPaint, Rect) <> 0 THEN BEGIN
|  |  |  |  |  |  |     Move (Info, PtInfo2, SizeOf (TPaintStruct));
|  |  |  |  |  |  |     PtInfo2.rcPaint := Inter;
|  |  |  |  |  |  |     Boite[I]^.Montre (DC, PtInfo2);
|  |  |  |  |  |  +--END;
|  |  |  |  |  +--END;
|  |  |  |  +--END;
|  |  |  +--END;
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.Quitter (VAR Msg: TMessage);
+--BEGIN
|     CloseWindow;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.SetUpWindow;
+--BEGIN
|     PostMessage (HWindow, wm_SysCommand, sc_Maximize, 0);
|     TWindow.SetUpWindow;
|  +--IF Gerant <> NIL THEN BEGIN
|  |     Gerant^.ModifieMenu;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.ClickBouton01 (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Gerant^.ClickBouton(1);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.ClickBouton02 (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Gerant^.ClickBouton(2);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.ClickBouton03 (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Gerant^.ClickBouton(3);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.ClickBouton04 (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Gerant^.ClickBouton(4);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.ClickBouton05 (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Gerant^.ClickBouton(5);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.WMCancelMode (VAR Msg: TMessage);
+--BEGIN
|     ReleaseCapture;
|     Msg.lParam := Position;
|     IF FocusGauche <> 0 THEN WMlButtonUp (Msg);
|     IF FocusDroit  <> 0 THEN WMrButtonUp (Msg);
|     Msg.Message := 0;
+--END;
   {-------------------------------------------------------------------------}
```

OBCaptur.pas

```
   UNIT OBCaptur;

   {-------------------------------------------------------------------------}
            INTERFACE
   {-------------------------------------------------------------------------}

   USES OWindows, WinTypes, WinProcs, Strings,
        Globaux, OBCases, ODidact;

   TYPE
   {-------------------------------------------------------------------------
        Objet TBoiteCases
   -------------------------------------------------------------------------}

   PBoiteCapture = ^TBoiteCapture;
   TBoiteCapture = OBJECT (TBoiteCases)
      OKCapture  : TSetByte;
      Curseur    : TRect;
      Contour    : {HBrush;}HPen;
      constructor Init
                  (UnParent: PWindow; UnTitre: PChar; TitreBis: Boolean;
                  AX, AY: Integer; NbX, NbY, DimX, DimY: Byte);
      destructor  Done;       virtual;

      procedure   CadreXOR       (Rect1, Rect2: TRect);
      procedure   MessageGerant (ID: Word; NoCase: Integer; Code: Word); virtual;
      procedure   lButtonDown    (var Msg: TMessage); virtual;
      procedure   MouseMove      (var Msg: TMessage); virtual;
      procedure   lButtonUp      (var Msg: TMessage); virtual;
      End;

   {-------------------------------------------------------------------------}
            IMPLEMENTATION
   {-------------------------------------------------------------------------}

   CONST
   NilRect : TRect = (Left:0; Top:0; Right:0; Bottom:0);

   CONSTRUCTOR TBoiteCapture.Init
      (UnParent: PWindow; UnTitre: PChar; TitreBis: BOOLEAN;
      AX, AY: INTEGER; NbX, NbY, DimX, DimY: Byte);
   BEGIN
      TBoiteCases.Init (UnParent, UnTitre, TitreBis, AX, AY, NbX, NbY, DimX, DimY);
      OKCapture := [];
      MetAZero (Curseur, SizeOf(Curseur));
      Contour := {CreateSolidBrush (GrisFonce);}
         CreatePen (ps_Solid, 1, GrisFonce);
   END;
   {-------------------------------------------------------------------------}
   DESTRUCTOR TBoiteCapture.Done;
+--BEGIN
|     TBoiteCases.Done;
|     DeleteObject (Contour);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoiteCapture.CadreXOR (Rect1, Rect2: TRect);
   VAR UnDC : HDC;
+--BEGIN
|     PrendDC (UnDC);
|     SetRop2 (UnDC, r2_XORPen);
|     SelectObject (UnDC, Contour);
|  {   DrawFocusRect (UnDC, Rect1);
|     DrawFocusRect (UnDC, Rect2); }
|     WITH Rect1 DO Rectangle (UnDC, Left, Top, Right, Bottom);
|     WITH Rect2 DO Rectangle (UnDC, Left, Top, Right, Bottom);
|     LacheDC (UnDC);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoiteCapture.MessageGerant (ID: Word; NoCase: INTEGER; Code: Word);
+--BEGIN
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoiteCapture.lButtonDown (VAR Msg: TMessage);
```

```
   VAR NoCase : INTEGER;
       Rect   : TRect;
+--BEGIN
|     NoCase := CasePointee (Msg);
|     IF NoCase IN OKCapture
|  +--THEN BEGIN
|  |       CoordCase (CasePointee(Msg), Rect);
|  |       Curseur := Rect;
|  |       CadreXOR (Rect, NilRect);
|  |       FlButton := fl_Prend;
|  |       MessageGerant (mg_Prend, NoCase, rr_SansErreur);
|  +-------END
|  +--ELSE BEGIN
|  |       IF NoCase <> -1 THEN MessageGerant (mg_Prend, NoCase, rr_NoCase);
|  |       FlButton := fl_Sans;
|  +-------END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoiteCapture.lButtonUp (VAR Msg: TMessage);
+--BEGIN
|  +--IF FlButton <> fl_Sans THEN BEGIN
|  |     CadreXOR (Curseur, NilRect);
|  |     MetAZero (Curseur, SizeOf(Curseur));
|  |     MessageGerant (mg_Depose, -1, rr_SansErreur);
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TBoiteCapture.MouseMove(VAR Msg: TMessage);
   VAR Rect: TRect;
+--BEGIN
|     IF FlButton IN [fl_Prend, fl_Deplace]
|  +--THEN BEGIN
|  |    +--WITH Rect DO BEGIN
|  |    |     Left   := Msg.lParamLo - DimCase.X DIV 2;
|  |    |     Right  := Left + DimCase.X - 1;
|  |    |     Top    := Msg.lParamHi - DimCase.Y DIV 2;
|  |    |     Bottom := Top + DimCase.Y - 1;
|  |    +--END;
|  |       CadreXOR (Curseur, Rect);
|  |       Curseur := Rect;
|  |       FlButton := fl_Deplace;
|  +-------END;
+--END;

   {-------------------------------------------------------------------------
            INITIALISATION
   -------------------------------------------------------------------------}
   END.
```

```
   PROCEDURE TFenetreVide.WMNClButtonUp (VAR Msg: TMessage);
+--BEGIN
|     ReleaseCapture;
|     IF FocusGauche <> 0 THEN WMlButtonUp (Msg);
|     IF FocusDroit  <> 0 THEN WMrButtonUp (Msg);
|     Msg.Message := 0;
+--END;
   {--------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.WMNCrButtonUp (VAR Msg: TMessage);
+--BEGIN
|     ReleaseCapture;
|     IF FocusGauche <> 0 THEN WMlButtonUp (Msg);
|     IF FocusDroit  <> 0 THEN WMrButtonUp (Msg);
|     Msg.Message := 0;
+--END;
   {--------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.WMlButtonDown (VAR Msg: TMessage);
   VAR I : Byte;
       Reponse : BOOLEAN;
+--BEGIN
|     SetCapture (HWindow);
|     IF (Msg.wParam AND MK_Shift) = MK_Shift
|     THEN WMlButtonShift (Msg)
|  +--ELSE BEGIN
|  |       Position := Msg.lParam;
|  |       I := 1;  Reponse := FALSE;
|  |    +--WHILE (I <= NBBoites) AND NOT Reponse DO BEGIN
|  |    |  +--WITH Boite[I]^ DO BEGIN
|  |    |  |     Reponse := Contient (Msg);
|  |    |  |  +--IF Reponse AND Peinte AND Active THEN BEGIN
|  |    |  |  |     FocusGauche := I;
|  |    |  |  |     lButtonDown (Msg);
|  |    |  |  +--END;
|  |    |  +--END;
|  |    |     Inc (I);
|  |    +--END;
|  +-------END;
+--END;
   {--------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.WMlButtonDblClk (VAR Msg: TMessage);
   VAR I : Byte;
       Reponse : BOOLEAN;
+--BEGIN
|     SetCapture (HWindow);
|     IF (Msg.wParam AND MK_Shift) = MK_Shift
|     THEN WMlButtonShift (Msg)
|  +--ELSE BEGIN
|  |       Position := Msg.lParam;
|  |       I := 1;  Reponse := FALSE;
|  |    +--WHILE (I <= NBBoites) AND NOT Reponse DO BEGIN
|  |    |  +--WITH Boite[I]^ DO BEGIN
|  |    |  |     Reponse := Contient (Msg);
|  |    |  |  +--IF Reponse AND Peinte AND Active THEN BEGIN
|  |    |  |  |     FocusGauche := I;
|  |    |  |  |     lButtonDblClk (Msg);
|  |    |  |  +--END;
|  |    |  +--END;
|  |    |     Inc (I);
|  |    +--END;
|  +-------END;
+--END;
   {--------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.WMMouseMove (VAR Msg: TMessage);
   VAR I : Byte;
+--BEGIN
|     Position := Msg.lParam;
|     IF FocusGauche <> 0 THEN Boite[FocusGauche]^.MouseMove (Msg);
|     IF FocusDroit  <> 0 THEN Boite[FocusDroit]^.MouseMove (Msg);
+--END;
   {--------------------------------------------------------------------------}
   PROCEDURE TFenetreVide.WMlButtonUp (VAR Msg: TMessage);
   VAR I : Byte;
       Reponse : BOOLEAN;
+--BEGIN
|     ReleaseCapture;
|     Position := Msg.lParam;
```

```
|   +--IF FocusGauche <> 0 THEN BEGIN
|   |   +--IF Boite[FocusGauche]^.FlButton IN [fl_Prend, fl_Deplace] THEN BEGIN
|   |   |       I := 1;  Reponse := FALSE;
|   |   |   +--WHILE (I <= NBBoites) AND NOT Reponse DO BEGIN
|   |   |   |       IF I <> FocusGauche THEN
|   |   |   |   +--WITH Boite[I]^ DO BEGIN
|   |   |   |   |       Reponse := Contient (Msg);
|   |   |   |   |   +--IF Reponse THEN BEGIN
|   |   |   |   |   |       lButtonUp (Msg);
|   |   |   |   |   |       FlButton := fl_Sans;
|   |   |   |   |   +--END;
|   |   |   |   +--END;
|   |   |   |       Inc (I);
|   |   |   +--END;
|   |   +--END;
|   |       I := FocusGauche;
|   |       FocusGauche := 0;
|   |   +--WITH Boite[I]^ DO BEGIN
|   |   |       lButtonUp (Msg);
|   |   |       FlButton := fl_Sans;
|   |   +--END;
|   +--END;
+--END;
    {-------------------------------------------------------------------------}
    PROCEDURE TFenetreVide.WMlButtonShift (VAR Msg: TMessage);
    VAR I : Byte;
        Reponse : BOOLEAN;
+--BEGIN
|       SetCapture (HWindow);
|       Position := Msg.lParam;
|       I := 1;  Reponse := FALSE;
|   +--WHILE (I <= NBBoites) AND NOT Reponse DO BEGIN
|   |   +--WITH Boite[I]^ DO BEGIN
|   |   |       Reponse := Contient (Msg);
|   |   |     +--IF Reponse AND Peinte AND Active THEN BEGIN
|   |   |     |       FocusGauche := I;
|   |   |     |       lButtonShift (Msg);
|   |   |     +--END;
|   |   +--END;
|   |       Inc (I);
|   +--END;
+--END;
    {-------------------------------------------------------------------------}
    PROCEDURE TFenetreVide.WMrButtonDown (VAR Msg: TMessage);
    VAR I : Byte;
        Reponse : BOOLEAN;
+--BEGIN
|       SetCapture (HWindow);
|       Position := Msg.lParam;
|       I := 1;  Reponse := FALSE;
|   +--WHILE (I <= NBBoites) AND NOT Reponse DO BEGIN
|   |   +--WITH Boite[I]^ DO BEGIN
|   |   |       Reponse := Contient (Msg);
|   |   |   +--IF Reponse AND Peinte AND Active THEN BEGIN
|   |   |   |       FocusDroit := I;
|   |   |   |       rButtonDown (Msg);
|   |   |   +--END;
|   |   +--END;
|   |       Inc (I);
|   +--END;
+--END;
    {-------------------------------------------------------------------------}
    PROCEDURE TFenetreVide.WMrButtonUp (VAR Msg: TMessage);
    VAR I : Byte;
        Reponse : BOOLEAN;
+--BEGIN
|       ReleaseCapture;
|       Position := Msg.lParam;
|   +--IF FocusDroit <> 0 THEN BEGIN
|   |       I := FocusDroit;
|   |       FocusDroit := 0;
|   |       Boite[I]^.rButtonUp (Msg);
|   +--END;
+--END;

    {-------------------------------------------------------------------------
```

```
      TGerantVide
   ------------------------------------------------------------------------}

   CONSTRUCTOR TGerantVide.Init;
+--BEGIN
|     Fenetre := PFenetreVide (Application^.MainWindow);
|     GarnitMainWindow;
|     TGerantVide.GarnitMainWindow;
|     ChangeParametres;
|     Stop := FALSE;
+--END;
   {------------------------------------------------------------------------}
   DESTRUCTOR TGerantVide.Done;
+--BEGIN
|     VideMainWindow;
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.AideExercice;
+--BEGIN
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.Avertissement (ID: INTEGER);
   VAR Chaine : TCommentaire;
       Titre  : ARRAY[1..20] OF CHAR;
+--BEGIN
|     StrPCopy (@Titre, 'Attention');
|     LoadString (HInstance, ID, @Chaine, MaxComment);
|     Fenetre^.Message (@Titre, @Chaine);
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.ChangeParametres;
+--BEGIN
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.ClickBouton (No: Byte);
+--BEGIN
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.DoneExercice;
+--BEGIN
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.GarnitMainWindow;
   VAR Rect : TRect;
       I : Byte;
+--BEGIN
|  +--WITH Rect DO BEGIN
|  |     Left := 0;
|  |     Top  := 0;
|  |     Right := GetSystemMetrics (sm_cxFullScreen);
|  |     Bottom:= GetSystemMetrics (sm_cyFullScreen);
|  +--END;
|     InvalidateRect (Fenetre^.HWindow, @Rect, TRUE);
|     WITH Fenetre^ DO
|     FOR I := 1 TO NBBoutons DO
|         IF Bouton[I] <> NIL THEN Bouton[I]^.Create;
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.InitExercice (Param: Pointer);
+--BEGIN
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.ModifieMenu;
   VAR IDMenu : HMenu;
+--BEGIN
|     IDMenu := GetMenu(Application^.MainWindow^.HWindow);
|     EnableMenuItem (IDMenu, cm_Choisir, mf_ByCommand OR mf_Grayed);
|     EnableMenuItem (IDMenu, cm_AideEx, mf_ByCommand OR mf_Grayed);
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.RecoitMessage (VAR Msg: TMsgDidact);
+--BEGIN
+--END;
   {------------------------------------------------------------------------}
   PROCEDURE TGerantVide.VideMainWindow;
   VAR Ind : INTEGER;
```

```
+--BEGIN
|  +--WITH PFenetreVide(Application^.MainWindow)^ DO BEGIN
|  |     FOR Ind := 1 TO NBBoites DO
|  |      +--IF Boite[Ind] <> NIL THEN BEGIN
|  |      |     Dispose (Boite[Ind], Done);
|  |      |     Boite[Ind] := NIL;
|  |      +--END;
|  |     NBBoites := 0;
|  |
|  |     FOR Ind := 1 TO NBBoutons DO
|  |      +--IF Bouton[Ind] <> NIL THEN BEGIN
|  |      |     Dispose (Bouton[Ind], Done);
|  |      |     Bouton[Ind] := NIL;
|  |      +--END;
|  |     NBBoutons := 0;
|  +--END;
+--END;

   {-------------------------------------------------------------------------------
            INITIALISATION
   -------------------------------------------------------------------------------}
   END.
```

DValeurs.rc

```
DLGVALEURS DIALOG 114, 57, 143, 126
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Valeurs"
BEGIN
      CONTROL "", 101, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 99, 42, 21
12
      CONTROL "", 102, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 99, 64, 21
12
      CONTROL "Entre les deux nombres à additionner :", -1, "STATIC", SS_CENTER | WS_CHILD |
WS_VISIBLE, 16, 8, 112, 21
      CONTROL "Premier nombre", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 12, 45, 57, 8
      CONTROL "Second nombre", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 12, 66, 57, 8
      CONTROL "", -1, "STATIC", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE, 5, 31, 133, 55
      CONTROL "OK", 1, "BUTTON", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP,
8, 94, 37, 25
      CONTROL "Annuler", 2, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 97, 94,
37, 25
END
```

DValeurs.pas

```
   UNIT DValeurs;

   {-------------------------------------------------------------------------}
            INTERFACE
   {-------------------------------------------------------------------------}

   {$R DValeurs.RC}

   USES OWindows, WinTypes, WinProcs, WinDos, Strings, ODialogs, Globaux;

+--FUNCTION DialogueValeurs (Fenetre: PWindow;
|           VAR Val1: INTEGER; Op: CHAR; VAR Val2: INTEGER): BOOLEAN;
|
|  {-------------------------------------------------------------------------}
|           IMPLEMENTATION
|  {-------------------------------------------------------------------------}
|
|  CONST
|  MaxChiffres=   4;
|
|  TYPE
|  {-------------------------------------------------------------------------
|       Boîte de dialogue : état des différents éléments
|  -------------------------------------------------------------------------}
|  TBufferValeurs = RECORD
|     EValeur1 : ARRAY[0..MaxChiffres] OF CHAR;
|     EValeur2 : ARRAY[0..MaxChiffres] OF CHAR;
|     END;
|  {
|  PDlgValeurs = ^TDlgValeurs;
|  TDlgValeurs = OBJECT (TDialog)
|     Symbole: Char;
|     Valeur1: Integer;
|     Valeur2: Integer;
|     Buffer : TBufferValeurs;
|     constructor Init (AParent: PWindow; AName: PChar; Op: Char; ABuffer:TBufferValeurs);
|     function    CanClose: Boolean; virtual;
|     procedure   OK (var Msg: TMessage); virtual id_First + idOK;
|     END;
|  }
|  {-------------------------------------------------------------------------
|       TDlgValeurs
|  -------------------------------------------------------------------------}
|  CONSTRUCTOR TDlgValeurs.Init
|              (AParent: PWindow; AName: PChar; Op: CHAR; ABuffer: TBufferValeurs);
|  CONST ID_Valeur1 = 101;
|        ID_Valeur2 = 102;
|  VAR B1, B2 : PEdit;
+--BEGIN
|     TDialog.Init (AParent, AName);
```

```
|     IF Op IN ['+','-'] THEN Symbole := Op ELSE Symbole := '+';
|     B1 := New (PEdit, InitResource(@Self, ID_Valeur1, Succ(MaxChiffres)));
|     B1^.Validator := New (PRangeValidatorF, Init (1, 9999));
|     B2 := New (PEdit, InitResource(@Self, ID_Valeur2, Succ(MaxChiffres)));
|     B2^.Validator := New (PRangeValidatorF, Init (1, 9999));
|     Buffer := ABuffer;
|     TransferBuffer := @Buffer;
+--END;
  {-------------------------------------------------------------------------}
+--FUNCTION TDlgValeurs.CanClose: BOOLEAN;
|  VAR Code1     : INTEGER;
|      Code2     : INTEGER;
|      Valeur    : REAL;
|      Err1,Err2 : BOOLEAN;
|      ErrSomme  : BOOLEAN;
|      Erreur    : BOOLEAN;
|      SValeur1  : STRING[MaxChiffres];
|      SValeur2  : STRING[MaxChiffres];
+--BEGIN
|     TransferData (tf_GetData);
|     SValeur1 := StrPas (Buffer.EValeur1);
|     SValeur2 := StrPas (Buffer.EValeur2);
|     Val (SValeur1, Valeur, Code1);
|     Valeur1 := Trunc (Valeur);
|     Val (SValeur2, Valeur, Code2);
|     Valeur2 := Trunc (Valeur);
|     Err1 := Valeur1 = 0;
|     Err2 := Valeur2 = 0;
|     ErrSomme := (Symbole = '+') AND (Valeur1 + Valeur2 > 9999);
|     Erreur := ErrSomme OR Err1 OR Err2;
|     IF Err1 THEN
|        MessageBox (HWindow, 'Les nombres doivent être compris entre 1 et 9999...', '
|         Fermeture',
|        mb_OK OR mb_IconInformation) ELSE
|     IF Err2 THEN
|        MessageBox (HWindow, 'Les nombres doivent être compris entre 1 et 9999...', '
|         Fermeture',
|        mb_OK OR mb_IconInformation) ELSE
|     IF ErrSomme THEN
|        MessageBox (HWindow, 'Diminue un des deux nombres : la somme dépasse 9999...',
|         'Fermeture',
|        mb_OK OR mb_IconInformation);
|     CanClose := NOT Erreur;
+--END;
  {-------------------------------------------------------------------------}
+--PROCEDURE TDlgValeurs.OK (VAR Msg: TMessage);
+--BEGIN
|     TDialog.OK (Msg);
+--END;

  {-------------------------------------------------------------------------
     Gestion du buffer et des paramètres par la procédure appelante
  -------------------------------------------------------------------------}
+--FUNCTION DialogueValeurs (Fenetre: PWindow;
|           VAR Val1: INTEGER; Op: CHAR; VAR Val2: INTEGER): BOOLEAN;
|  VAR Dlg       : PDlgValeurs;
|      OldBuffer : TBufferValeurs;
|      Return    : INTEGER;
+--BEGIN
|     MetAZero (OldBuffer, SizeOf(OldBuffer));
|     Dlg := New (PDlgValeurs, Init (Fenetre, 'DlgValeurs', Op, OldBuffer));
|     Return := Dlg^.Execute;
|     IF (Return = IdOK)
|  +--THEN BEGIN
|  |       Val1 := Dlg^.Valeur1;
|  |       Val2 := Dlg^.Valeur2;
|  |       DialogueValeurs := TRUE;
|  +-------END
|     ELSE DialogueValeurs := FALSE;
|     Dispose (Dlg, Done);
+--END;
  {-------------------------------------------------------------------------}
  END.
```

SAdd001.rc

```
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      0, "Nous allons additionner deux nombres. Ils sont représentés en bonbons, mais aussi en
chiffres dans la boîte du calcul écrit. Pour faire le calcul, nous partons du rang le plus petit (1
bonbons) pour arriver au plus élevé."
      1, "Pour choisir deux nombres et les additionner, clique sur le bouton <Valeurs>.
\r\nEventuellement, consulte le menu Options | Générales | Qui choisit les valeurs ? ..."
      2, "Voilà, l'addition est terminée."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      17, "[VALEURS EN BONBONS] \r\nIl n'y a pas de bonbon. Regarde le calcul écrit : le résultat
pour les unités est 0."
      18, "[VALEURS EN BONBONS]  \r\nLes deux nombres ne comprennent aucun sachet. Cela fait une
somme de 0 au 2e rang."
      19, "[VALEURS EN BONBONS]  \r\nPuisqu'il n'y a pas de caisse, nous inscrivons 0 comme résulta
au 3e rang."
      20, "[VALEURS EN BONBONS]  \r\nLe nombre total d'armoires est 0."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      33, "[VALEURS EN BONBONS]  \r\nDans les deux nombres, sélectionne tous les objets de type
\042bonbon\042. Ils seront ensuite copiés pour effectuer l'addition."
      34, "[VALEURS EN BONBONS]  \r\nSélectionne tous les sachets des deux nombres. Ils seront
copiés pour effectuer l'addition."
      35, "[VALEURS EN BONBONS]  \r\nSélectionne toutes les caisses des deux nombres. Tu verras leu
copie s'afficher pour l'addition."
      36, "[VALEURS EN BONBONS]  \r\nDans les deux nombres, sélectionne toutes les armoires pour le
utiliser dans le rectangle d'addition."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      49, "[VALEURS EN BONBONS] \r\nTu peux copier les bonbons des deux nombres."
      50, "[VALEURS EN BONBONS] \r\nCopie à présent les sachets des deux nombres."
      51, "[VALEURS EN BONBONS] \r\nMaintenant, ce sont les caisses qui doivent être copiées."
      52, "[VALEURS EN BONBONS] \r\nIl ne reste que les armoires à copier."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      65, "[ADDITION] \r\nEn base 10, nous ne pouvons écrire que des chiffres de 0 à 9. Or, le
nombre de bonbons est supérieur à neuf. Regroupe dix bonbons en un sachet."
      66, "[ADDITION] \r\nOn ne peut pas écrire le nombre de sachets avec un seul chiffre. Regroupe
des sachets pour obtenir une caisse."
      67, "[ADDITION] \r\nLe nombre de caisses dépasse la base : il n'y a pas de chiffre pour écrir
le nombre de caisses. Regroupes-en pour obtenir un report."
      68, "[ADDITION] \r\nLà, il y a un problème..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      81, "[CALCUL ECRIT]\r\nBien ! Tu as regroupé les bonbons, un sachet apparaît.Dans le calcul,
le remplacement effectué se marque par un report de 1 au rang des dizaines (sachets)."
      82, "[CALCUL ECRIT] \r\nTu viens d'effectuer un report. L'apparition d'une caisse
supplémentaire est indiquée dans le calcul écrit : on écrit 1 au 3e rang."
      83, "[CALCUL ECRIT] \r\nTu as fais apparaître une nouvelle armoire, nous inscrivons 1 comme
report au rang des milliers."
      84, "[CALCUL ECRIT] \r\nLà, il y a un problème..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      97, "[ADDITION] \r\nSélectionne tous les bonbons pour faire apparaître le résultat au rang de
unités."
      98, "[ADDITION] \r\nSélectionne tous les sachets : tu feras ainsi apparaître le résultat de
l'addition au rang 2."
      99, "[ADDITION] \r\nSélectionne les caisses : le chiffre sera placé dans le calcul écrit au
rang des centaines."
      100, "[ADDITION] \r\nSélectionne les armoires pour compléter le résultat de l'addition."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      113, "[ADDITION] \r\nIl n'y a plus de bonbons. Nous inscrivons 0 comme résultat au rang 1."
      114, "[ADDITION] \r\nIl ne reste plus de sachet, cela fait 0 au rang des dizaines dans le
calcul écrit."
      115, "[ADDITION] \r\nComme il n'y a plus de caisse, le chiffre de la somme au rang 3 est 0."
```

```
      116, "[ADDITION] \r\nLà, il y a un problème..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      129, "[VALEURS EN BONBONS] \r\nLà, il y a un problème..."
      130, "[VALEURS EN BONBONS] \r\nLes deux nombres ne comprennent aucun sachet. \r\n[ADDITION]Pa
contre, le report que tu viens d'effectuer doit figurer comme résultat. Sélectionne l'unique sachet
pour passer au rang suivant."
      131, "[VALEURS EN BONBONS] \r\nPuisqu'il n'y a pas de caisse dans les deux nombres, seul le
report que tu as effectué compte pour l'addition. \r\n[ADDITION] Sélectionne donc la seule caisse
pour qu'elle apparaisse comme résultat."
      132, "[VALEURS EN BONBONS] \r\nIl n'y a pas d'armoire dans les deux nombres. Par contre, j'er
vois une apparue comme report. \r\n[ADDITION] Sélectionne l'armoire pour compléter l'addition au ra
des milliers."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      145, "[ADDITION] \r\nSi nécessaire, regroupe des bonbons en un sachet. Puis sélectionne les
bonbons restants comme résultat du rang."
      146, "[ADDITION] \r\nVois s'il faut regrouper des sachets en une caisse. Ensuite, fais
apparaître le résultat pour le rang des dizaines en sélectionnant tous les sachets restants."
      147, "[ADDITION] \r\nRegroupe des caisses s'il le faut. Ensuite, marque les caisses pour
obtenir le résultat au 3e rang."
      148, "[ADDITION] \r\nSélectionne les armoires pour compléter la somme au 4e rang."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      161, "Aucun bonbon ne s'affichera dans le rectangle d'addition tant qu'il ne seront pas tous
sélectionnés dans les deux nombres. Je vois encore des bonbons sur lesquels tu peux cliquer."
      162, "Tu as oublié des sachets dans les deux nombres à additionner. Tant que tu ne les auras
pas sélectionnés tous, ils ne s'afficheront pas dans le rectangle d'addition. "
      163, "Il reste des caisses non sélectionnées dans les deux nombres. Quand toutes le seront,
elles s'afficheront dans le rectangle d'addition. Clique sur les caisses non sélectionnées."
      164, "Il reste des armoires non sélectionnées dans les deux nombres. Clique dessus. Quand
toutes seront sélectionnées, alors tu pourras les utiliser dans le rectangle d'addition."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      177, "Nous n'utilisons que des nombres entiers. Donc, comme le bonbon est l'objet le plus
petit, tu ne peux pas l'ouvrir."
      178, "Au lieu d'un sachet, tu as 10 bonbons. Si tu les copiais pour l'addition, tu devrais le
regrouper  et tu obtiendrais le report d'un sachet. REGROUPE LES BONBONS MAINTENANT."
      179, "Au lieu d'une caisse, tu as 10 sachets. Si tu les copiais pour l'addition, tu devrais
quand même les regrouper en une caisse. Alors, regroupe les sachets maintenant."
      180, "Au lieu d'une armoire, tu as 10 caisses. Si tu les copiais pour l'addition, tu devrais
quand même les regrouper pour avoir de nouveau une armoire. Alors, regroupe-les maintenant."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      193, "Nous n'utilisons que des nombres entiers. Donc, comme le bonbon est l'objet le plus
petit, tu ne peux pas l'ouvrir"
      194, "Comme les chiffres vont de 0 à 9, tu ne peux pas avoir plus de 9 bonbons, sinon tu ne
peux pas écrire le nombre. Regroupe ces bonbons pour retrouver le sachet."
      195, "Comme les chiffres vont de 0 à 9, tu ne peux pas avoir plus de 9 sachets, sinon tu ne
peux pas écrire le nombre Regroupe ces sachets pour retrouver une caisse."
      196, "Utiliser une armoire, c'est plus pratique que 10 caisses. Et comme les chiffres vont de
0 à 9, tu ne peux pas écrire le nombre si tu as plus de 9 caisses. Regroupe ces caisses pour
retrouver une armoire."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      209, "Tu as déjà additionné les bonbons. Le résultat au unités ne changera pas."
      210, "Tu as déjà additionné les sachets. Le résulat aux dizaines ne changera pas."
      211, "Tu as déjà additionné les caisses. Le résultat aux centaines ne changera pas."
      212, "L'addition est terminée..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      225, "Pas trop vite..."
      226, "Tu pourrais calculer la somme des sachets avant celle des bonbons. Mais, après, si tu
avais 10 bonbons à regrouper en un sachet, tu devrais changer le résulat des dizaines."
      227, "Suppose que tu copies déjà les caisses et que tu calcules leur somme avant celle des
sachets. Si tu devais regrouper dix sachets en une caisse, il faudrait changer le résultat. "
      228, "Tu pourrais calculer la somme des armoires avant celle des autres objets. Mais, après,
si tu devais regrouper dix caisses en une armoire, tu devrais changer le résulat au 4e rang."
END
```

```
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	241, "S'il y a plus de 9 bonbons, tu en sélectionnes EXACTEMENT 10 pour faire un report. Si l
report est déjà fait, ou bien s'il n'y a pas plus de 9 bonbons, sélectionne-les tous."
	242, "S'il y a plus de 9 sachets, sélectionnes-en EXACTEMENT 10 pour faire un report. Si le
report est déjà fait, ou s'il ne faut pas en faire, sélectionne tous les sachets de l'addition pour
voir le résulat."
	243, "S'il y a plus de 9 caisses, sélectionnes-en EXACTEMENT 10 pour faire un report. Si le
report est déjà fait ou s'il ne faut pas en faire, sélectionne toutes les caisses pour passer au ra
suivant."
	244, "Quand tu auras sélectionné en même temps toutes les armoires de l'addition, tu auras
terminé."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	257, "Erreur..."
	258, "Dans le rectangle d'addition, il y a encore dix bonbons que tu peux regrouper en un
sachet. Fais-le avant toute autre opération."
	259, "Dans le rectangle d'addtion, tu peux regrouper dix sachets en une caisse. Fais-le avant
toute autre chose."
	260, "Dans le rectangle d'addition, tu as ouvert par erreur une armoire. Regroupe les caisses
pour retrouver l'armoire."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	273, "Tu avais bien commencé. Les bonbons ne seront copiés que quand tu les auras sélectionné
TOUS DANS LES DEUX NOMBRES."
	274, "C'était bien parti. Sélectionne les sachets. Ils ne seront copiés pour l'addition que
quand tu les auras TOUS sélectionnés DANS LES DEUX NOMBRES."
	275, "Tu peux cliquer sur les caisses dans n'importe quel ordre, mais tu dois les sélectionne
TOUTES DANS LES DEUX NOMBRES avant que je ne les copie pour l'addition."
	276, "Tu avais bien commencé. Sélectionne TOUTES les armoires DANS LES DEUX NOMBRES."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	289, "S'il y a plus de 9 bonbons, tu en sélectionnes EXACTEMENT 10 pour faire un report. Si l
report est déjà fait, ou bien s'il n'y a pas plus de 9 bonbons, sélectionne-les tous."
	290, "S'il y a plus de 9 sachets, sélectionnes-en EXACTEMENT 10 pour faire un report. Si le
report est déjà fait, ou s'il ne faut pas en faire, sélectionne tous les sachets de l'addition pour
voir le résulat."
	291, "S'il y a plus de 9 caisses, sélectionnes-en EXACTEMENT 10 pour faire un report. Si le
report est déjà fait ou s'il ne faut pas en faire, sélectionne toutes les caisses pour passer au ra
suivant."
	292, "Quand tu auras sélectionné en même temps toutes les caisses de l'addition, tu auras
terminé."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	305, "Erreur..."
	306, "Tu avais ouvert un sachet par erreur. Sélectionne exactement 10 bonbons pour les
regrouper en un sachet."
	307, "Tu avais ouvert une caisse par erreur. Sélectionne exactement 10 sachets pour les
regrouper en une caisse."
	308, "Tu avais ouvert une armoire par erreur. Sélectionne exactement 10 caisses pour les
regrouper."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	321, "Nous n'utilisons que des nombres entiers. Donc, comme le bonbon est l'objet le plus
petit, tu ne peux pas l'ouvrir."
	322, "Si tu ouvrais le sachet, tu aurais 10 bonbons de plus. Mais comme le plus grand chiffre
est 9, tu ne peux pas avoir plus de 9 bonbons, sinon tu ne peux pas écrire le nombre."
	323, "Si tu ouvrais la caisse, tu aurais 10 sachets de plus. Mais comme le plus grand chiffre
est 9, tu ne pourrais pas écrire le nombre."
	324, "Si tu ouvrais l'armoire, tu aurais 10 caisses de plus. Mais comme le plus grand chiffre
est 9, tu ne pourrais pas écrire le nombre."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	337, "Nous n'utilisons que des nombres entiers. Donc, comme le bonbon est l'objet le plus
petit, tu ne peux pas l'ouvrir."
	338, "Ce sachet représente dix bonbons. Si tu l'ouvrais, tu aurais 10 bonbons de plus. Mais
comme le plus grand chiffre est 9, tu ne peux pas avoir plus de 9 bonbons."
	339, "Cette caisse vaut dix sachets. Si tu l'ouvrais, tu aurais 10 sachets de plus. Mais comm
le plus grand chiffre est 9, tu ne pourrais pas écrire le nombre."
```

```
      340, "Cette amoires vaut dix caisses. Si tu l'ouvrais, tu aurais 10 caisses de plus. Mais
comme le plus grand chiffre est 9, tu ne pourrais pas écrire le nombre."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      353, "Pour regrouper des bonbons en un sachet, tu dois en sélectionner EXACTEMENT 10. Là, ça
fait plus que 10."
      354, "Pour regrouper des sachets en une caisse, tu dois en sélectionner EXACTEMENT 10. Là, ça
fait plus que 10."
      355, "Pour regrouper des caisses en une armoire, tu dois en sélectionner EXACTEMENT 10. Là, ç
fait plus que 10."
      356, "Tralalère. Pas plus de 10."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      369, "Tu as déjà copié tous les bonbons. Maintenant, tu dois travailler dans le rectangle
d'addition. Pour faire un report, sélectionne EXACTEMENT 10 bonbons. Si le report est déjà fait, ou
bien s'il n'y a pas plus de 9 bonbons, sélectionne-les tous."
      370, "Tu as déjà copié tous les sachets. Maintenant, tu dois travailler dans le rectangle
d'addition.  Pour faire un report, sélectionne EXACTEMENT 10 sachets. Si le report est déjà fait, o
bien s'il n'y a pas plus de 9 sachets, sélectionne-les tous."
      371, "Tu as déjà copié toutes les caisses. Maintenant, tu dois travailler dans le rectangle
d'addition.  Pour faire un report, sélectionne EXACTEMENT 10 caisses. Si le report est déjà fait, o
bien s'il n'y a pas plus de 9 caisses, sélectionne-les toutes."
      372, "Tu as déjà copié toutes les armoires. Maintenant, tu dois travailler dans le rectangle
d'addition.  Pour faire un report, sélectionne EXACTEMENT 10 caisses. Si le report est déjà fait, o
bien s'il n'y a pas plus de 9 caisses, sélectionne-les toutes."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
      385, "Pour faire apparaître le résultat au rang des unités, sélectionne tous les bonbons
restants."
      386, "Pour faire apparaître le résultat au rang des dizaines, sélectionne tous les sachets
restants."
      387, "Pour faire apparaître le résultat au rang des centaine, sélectionne toutes les caisses
restantes."
      388, "Pour faire apparaître le résultat au rang des milliers, sélectionne toutes les armoires
restantes."
END
```

GAdd001.pas

```
UNIT GAdd001;

{$R SAdd001.RC}

{-------------------------------------------------------------------------}
        INTERFACE
{-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, Strings,
     Globaux, DValeurs, ODidact, OCalcAS,
     OBoite, OBIcones, OBTexte, OBAddSub, OBouton;

TYPE
{-------------------------------------------------------------------------
     TGerantAdd001
-------------------------------------------------------------------------}

PGerantAdd001 = ^TGerantAdd001;
TGerantAdd001 = OBJECT (TGerantVide)
   BNombre     : Array[1..2] of PBIcones;
   BSomme      : PBIcones;
   BCalcul     : PBAddSub;
   BExplic     : PBTExte;
   BValeurs    : PBouton;

   Calcul      : TCalcAddSub;
   RangCourant : VObjet;
   RangTermine : Array[VObjet] of Boolean;
   ToutAbaisse : Array[VObjet] of Boolean;
   ReportFait  : Array[VObjet] of Boolean;
   OuvertErreur: Array[VObjet] of Boolean;
   AddTerminee : Boolean;
   IndicAction : Boolean;

   constructor Init;
   destructor  Done;              virtual;

   procedure   ChangeParametres; virtual;
   procedure   ChoisitValeurs    (var Val1, Val2: Integer; var OK: Boolean);
   procedure   ClickBouton       (No: Byte); virtual;
   procedure   Commentaire       (ID1, ID2 : Integer);
   procedure   DoneExercice;     virtual;
   procedure   DoneRang;
   procedure   GarnitMainWindow; virtual;
   procedure   InitExercice      (Param: Pointer); virtual;
   procedure   InitRang;
   function    Ouverture:        VObjet;
   procedure   RecoitMessage     (var Msg: TMsgDidact); virtual;

   procedure   Recoit_MG_Select  (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Recoit_MG_Deselect(IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Recoit_MG_Ouvre   (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Recoit_MG_Ferme   (IndOrig: Byte; var Msg: TMsgDidact);

   procedure   Erreur_MG_Select  (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Erreur_MG_Deselect(IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Erreur_MG_Ouvre   (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Erreur_MG_Ferme   (IndOrig: Byte; var Msg: TMsgDidact);
   END;

{-------------------------------------------------------------------------}
        IMPLEMENTATION
{-------------------------------------------------------------------------}

CONST
{-------------------------------------------------------------------------
     Identificateurs des chaînes de caractères
          cg_XXXX : instructions pour effectuer l'exercice
          ag_XXXX : messages lors d'une mauvaise manipulation
-------------------------------------------------------------------------}
cg_Rien         = -0001;
cg_Inchange     = -0002;
```

```
   cg_Debut         = 0000;
   cg_Valeurs       = 0001;
   cg_Fin           = 0002;
   cg_YaPasDe       = 0016;
   cg_CopieTout     = 0032;
   cg_CopieTout2    = 0048;
   cg_FaisReport    = 0064;
   cg_ReportFait    = 0080;
   cg_SelectReste   = 0096;
   cg_YaPlusDe      = 0112;
   cg_ReportSeul    = 0128;
   cg_TraiteRang    = 0144;


   ag_PasEncoreToutAbaisse    = 0160;
   ag_OuvertureInutileNombre = 0176;
   ag_OuvertureInutileSomme   = 0192;
   ag_RangDejaFait            = 0208;
   ag_RangPasEncore           = 0224;
   ag_TraiteRang              = 0240;
   ag_PasEncoreReferme        = 0256;
   ag_PourquoiDeselectNombre = 0272;
   ag_PourquoiDeselectSomme   = 0288;
   ag_OKFermeture             = 0304;
   ag_RefusOuvertureNombre    = 0320;
   ag_RefusOuvertureSomme     = 0336;
   ag_TropPourFermer          = 0352;
   ag_NombresDejaCopies       = 0368;
   ag_PasEncoreResultat       = 0384;

   {-------------------------------------------------------------------------------
       TGerantAdd001
   -------------------------------------------------------------------------------}

   CONSTRUCTOR TGerantAdd001.Init;
   BEGIN
      TGerantVide.Init;
      Fenetre^.SetCaption ('Addition en bonbons');
      AddTerminee := TRUE;
      Commentaire (CG_Debut, CG_Rien);
   END;
   {-------------------------------------------------------------------------------}
   DESTRUCTOR TGerantAdd001.Done;
+--BEGIN
|     VideMainWindow;
+--END;
   {-------------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.ChangeParametres;
+--BEGIN
|     IndicAction := Fenetre^.Parametres.Guide = gd_Detaille;
+--END;
   {-------------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.ClickBouton (No: Byte);
+--BEGIN
|     IF No = 1 THEN InitExercice (NIL);
+--END;
   {-------------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.ChoisitValeurs (VAR Val1, Val2: INTEGER; VAR OK: BOOLEAN);
   VAR But : INTEGER;
+--BEGIN
|  +--CASE Fenetre^.Parametres.Choix OF
|  |+-ch_Eleve : BEGIN
|  ||    OK := DialogueValeurs (Fenetre, Val1, '+', Val2);
|  |+----END;
|  |+-ch_Professeur : BEGIN
|  ||    Randomize;
|  || +--CASE Fenetre^.Parametres.NBRangs OF
|  || |  3 : But := 2 + Random (998);
|  || |  4 : But := 2 + Random(9998);
|  || +--END;
|  ||    Val1 := 1 + Random(But);
|  ||    Val2 := But - Val1;
|  ||    OK := TRUE;
|  |+----END;
|  |+-ch_Ordinateur : BEGIN
|  ||    Randomize;
```

```
|  || +--CASE Fenetre^.Parametres.NBRangs OF
|  || |  3 : But := 2 + Random (998);
|  || |  4 : But := 2 + Random(9998);
|  || +--END;
|  ||    Val1 := 1 + Random(But);
|  ||    Val2 := But - Val1;
|  ||    OK := TRUE;
|  |+----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Commentaire (ID1, ID2 : INTEGER);
   VAR Buffer1 : TCommentaire;
       Buffer2 : TCommentaire;
       NL      : ARRAY[1..3] OF CHAR;
+--BEGIN
|     MetAZero (Buffer1, SizeOf (Buffer1));
|     MetAZero (Buffer2, SizeOf (Buffer2));
|     IF ID1 <> cg_Rien THEN LoadString (HInstance, ID1, @Buffer1, MaxComment);
|     IF ID2 <> cg_Rien THEN LoadString (HInstance, ID2, @Buffer2, MaxComment);
|     NL := #$0D#$0A#$00;
|     StrLCat (@Buffer1, @NL, MaxComment);
|     StrLCat (@Buffer1, @Buffer2, MaxComment);
|     BExplic^.PrendTexte (Buffer1);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.DoneExercice;
+--BEGIN
|     Commentaire (CG_Fin, cg_Valeurs);
|     AddTerminee := TRUE;
|     Stop := TRUE;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.DoneRang;
   VAR I : Byte;
+--BEGIN
|     BSomme^.VerrouilleObjet (RangCourant);
|     BCalcul^.SelectionneRang (RangCourant, FALSE);
|     IF RangCourant = Calcul.RangMax
|  +--THEN BEGIN
|  |       DoneExercice;
|  +-------END
|  +--ELSE BEGIN
|  |       Inc (RangCourant);
|  |       InitRang;
|  +-------END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.GarnitMainWindow;
   VAR Max     : TPoint;
       DimBIc  : TPoint;
       DimBIc2 : TPoint;
       BordFen : TPoint;
       X, Y    : ARRAY[1..8] OF INTEGER;
       Texte   : ARRAY[0..20] OF CHAR;
+--BEGIN
|     Max.X := GetSystemMetrics (sm_cxFullScreen);
|     Max.Y := GetSystemMetrics (sm_cyFullScreen) - GetSystemMetrics (sm_cyMenu);
|     DimBIc.Y  := 2*(HauteurTitre + BordIntBoite) + 4*(HauteurIcone + EntreIcones);
|     DimBIc.X  := 2*(BordIntBoite + 1) +  9*(LargeurIcone + EntreIcones);
|     DimBIc2.X := 2*(BordIntBoite + 1) + 19*(LargeurIcone + EntreIcones);
|     BordFen.X := (Max.X - DimBIc2.X) DIV 4;
|     BordFen.Y := (Max.Y - 3*DimBIc.Y) DIV 4;
|     X[1] := BordFen.X;                  Y[1] := BordFen.Y;
|     X[2] := X[1];                       Y[2] := Y[1] + DimBIc.Y + BordFen.Y;
|     X[3] := (Max.X - DimBIc2.X) DIV 2;  Y[3] := Y[2] + DimBIc.Y + BordFen.Y;
|
|     BNombre[1] := New (PBIcones, Init(Fenetre,'Première valeur',X[1],Y[1],9,4));
|     Fenetre^.AjouteBoite (BNombre[1]);
|     BNombre[2] := New (PBIcones, Init(Fenetre,'Seconde valeur',X[2],Y[2],9,4));
|     Fenetre^.AjouteBoite (BNombre[2]);
|     BSomme := New (PBIcones, Init(Fenetre,'Addition',X[3],Y[3],19,4));
|     Fenetre^.AjouteBoite (BSomme);
|
|     X[4] := BNombre[1]^.X2 + BordFen.X;
|     Y[4] := (Y[3] - (2*(HauteurTitre+BordIntBoite)+5*20)) DIV 2;
```

```
|     BCalcul := New (PBAddSub, Init(Fenetre,'Calcul écrit',X[4],Y[4]));
|     Fenetre^.AjouteBoite (BCalcul);
|
|     X[5] := BCalcul^.X2 + BordFen.X;   Y[5] := BordFen.Y;
|     X[6] := Max.X - BordFen.X;         Y[6] := Y[3] - BordFen.Y;
|     BExplic := New (PBTexte,
|                Init(Fenetre,'Commentaires',X[5],Y[5],Max.X-BordFen.X,Y[3]-BordFen.Y));
|     Fenetre^.AjouteBoite (BExplic);
|
|     StrLCopy (@Texte, 'Nouv. valeurs', 20);
|     TailleBouton (Fenetre, @Texte, X[8], Y[8]);
|     X[7] := BNombre[1]^.X2 + (BExplic^.X1-BNombre[1]^.X2-X[8]) DIV 2;
|     Y[7] := BCalcul^.Y2 + 10;
|     Fenetre^.AjouteBouton (BValeurs,Texte, X[7],Y[7],X[7]+X[8],Y[7]+Y[8]);
|     BValeurs^.DevientDefaut;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.InitExercice (Param: Pointer);
   VAR Val  : ARRAY[1..2] OF INTEGER;
       Rang : VObjet;
       I    : INTEGER;
       OK   : BOOLEAN;
+--BEGIN
|     Calcul.Base := 10;
|     IF Param = NIL
|     THEN ChoisitValeurs (Val[1], Val[2], OK)
|  +--ELSE BEGIN
|  |       Move (Param^, Val, 4);
|  |       Val[1] := Abs(Val[1]);
|  |       Val[2] := Abs(Val[2]);
|  |       OK := Val[1] + Val[2] <= 9999;
|  +-------END;
|
|  +--IF OK THEN BEGIN
|  |     FOR i := 1 TO 2 DO
|  |  +--WITH BNombre[i]^ DO BEGIN
|  |  |     MonoSelect := TRUE;
|  |  |     OKOuvrir   := [];
|  |  |     OKFermer   := [];
|  |  |     OKSelect   := [];
|  |  |     OKDeselect := [];
|  |  |     PrendValeur (Val[i],Calcul.Base);
|  |  +--END;
|  |
|  |  +--WITH BSomme^ DO BEGIN
|  |  |     MonoSelect := TRUE;
|  |  |     OKOuvrir   := [];
|  |  |     OKFermer   := [];
|  |  |     OKSelect   := [];
|  |  |     OKDeselect := [];
|  |  |     PrendValeur (0, Calcul.Base);
|  |  +--END;
|  |     BCalcul^.PrendValeurs (Val[1], '+', Val[2], Calcul.Base);
|  |     Calcul.PrendValeurs (Val[1], '+', Val[2], Calcul.Base);
|  |
|  |     IndicAction := PFenetreVide(Application^.MainWindow)^.Parametres.Guide = gd_Detaille;
|  |     MetAZero (RangTermine, SizeOf (RangTermine));
|  |     MetAZero (ToutAbaisse, SizeOf (ToutAbaisse));
|  |     MetAZero (OuvertErreur, SizeOf (OuvertErreur));
|  |     MetAZero (ReportFait, SizeOf (ReportFait));
|  |     RangCourant := Bonbon;
|  |     AddTerminee := FALSE;
|  |     Stop        := FALSE;
|  |     InitRang;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.InitRang;
   VAR i : INTEGER;
       N : INTEGER;
+--BEGIN
|+-WITH Calcul DO BEGIN
||    FOR i := 1 TO 2 DO BNombre[i]^.OKSelect := [RangCourant];
|| +--WITH BSomme^ DO BEGIN
|| |     OKSelect   := [Bonbon..Armoire];
|| |     OKDeselect := [Bonbon..Armoire];
```

```
|| |     IF RangCourant > Bonbon  THEN OKOuvrir := OKOuvrir + [RangCourant];
|| |     IF RangCourant < Armoire THEN OKFermer := OKFermer + [RangCourant];
|| +--END;
||
||    BCalcul^.SelectionneRang (RangCourant, TRUE);
||
||    IF ResultatBrutOK[RangCourant] = 0
|| +--THEN BEGIN
|| |       ToutAbaisse[RangCourant] := TRUE;
|| |       IF ResultatOK[RangCourant] = 0
|| |    +--THEN BEGIN
|| |    |       IF IndicAction THEN
|| |    |          Avertissement (CG_YaPasDe + Ord(RangCourant));
|| |    |       BCalcul^.PrendResultat (RangCourant, 0);
|| |    |       DoneRang;
|| |    +-------END
|| |       ELSE Commentaire (CG_ReportSeul + Ord(RangCourant), CG_Rien);
|| +-------END
||    ELSE Commentaire (CG_CopieTout + Ord(RangCourant), CG_Rien);
|+-END;
+--END;
   {-------------------------------------------------------------------------}
   FUNCTION TGerantAdd001.Ouverture: VObjet;
   VAR Objet : Byte;
       Resultat : VObjet;
+--BEGIN
|     Resultat := Rien;
|     Objet := 0;
|  +--WHILE (Resultat = Rien) AND (Objet <= Ord(Armoire)) DO BEGIN
|  |     IF OuvertErreur[ObjetDeRang[Objet]]
|  |  +--THEN BEGIN
|  |  |       Resultat := ObjetDeRang[Objet];
|  |  |       Objet := Succ(Ord(Armoire));
|  |  +-------END
|  |     ELSE Inc (Objet);
|  +--END;
|     Ouverture := Resultat;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Recoit_MG_Select (IndOrig: Byte; VAR Msg: TMsgDidact);
   VAR Message : TMessage;
+--BEGIN
|  +--WITH Calcul, Msg DO CASE IndOrig OF
|  |+-1,2: BEGIN
|  ||       ToutAbaisse[RangCourant] :=
|  ||          (BNombre[1]^.NbSelect[RangCourant,Total] +
|  ||           BNombre[2]^.NbSelect[RangCourant,Total]) =
|  ||          (BNombre[1]^.Card[Total,RangCourant] +
|  ||           BNombre[2]^.Card[Total,RangCourant]);
|  ||    +--IF ToutAbaisse[RangCourant] THEN BEGIN
|  ||    |     BNombre[1]^.VerrouilleObjet (RangCourant);
|  ||    |     BNombre[2]^.VerrouilleObjet (RangCourant);
|  ||    |  +--WITH BSomme^ DO BEGIN
|  ||    |  |     Inc (Card[NPos][RangCourant], ResultatBrutOK[RangCourant]);
|  ||    |  |     PrendCardinal (Card, Base);
|  ||    |  +--END;
|  ||    |     IF IndicAction
|  ||    |  +--THEN BEGIN
|  ||    |  |       IF ReportOKDe[RangCourant]
|  ||    |  |       THEN Commentaire (CG_FaisReport + Ord(RangCourant), CG_Rien)
|  ||    |  |       ELSE Commentaire (CG_SelectReste+ Ord(RangCourant), CG_Rien);
|  ||    |  +-------END
|  ||    |  +--ELSE BEGIN
|  ||    |  |       Commentaire (CG_TraiteRang + Ord(RangCourant), CG_Rien);
|  ||    |  +-------END;
|  ||    +--END;
|  |+------END;
|  |+-3, 4, 5 : BEGIN
|  ||       IF OuvertErreur[RangCourant]
|  ||    +--THEN BEGIN
|  ||    +-------END
|  ||    +--ELSE BEGIN
|  ||    |       IF NOT ToutAbaisse[RangCourant]
|  ||    |       THEN Avertissement (ag_PasEncoreToutAbaisse + Ord(RangCourant));
|  ||    |       IF (ReportOKDe[RangCourant] AND ReportFait[RangCourant]) OR
|  ||    |          (NOT ReportOKDe[RangCourant])
```

```
|  ||   |     +--THEN BEGIN
|  ||   |     |       IF (BSomme^.Card[Total][RangCourant] = ResultatOK[RangCourant]) AND
|  ||   |     |          (BSomme^.NbSelect[RangCourant,Total] = ResultatOK[RangCourant])
|  ||   |     |     +--THEN BEGIN
|  ||   |     |     |       BCalcul^.PrendResultat (RangCourant, ResultatOK[RangCourant]);
|  ||   |     |     |       DoneRang;
|  ||   |     |     +-------END;
|  ||   |     +-------END;
|  ||   +-------END;
|  |+------END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Recoit_MG_Deselect (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|  +--WITH Msg DO BEGIN
|  |  +--CASE IndOrig OF
|  |  |  1, 2 : Avertissement (ag_PourquoiDeselectNombre + Ord(MObjet));
|  |  |  3    : IF Ouverture <> Rien
|  |  |       +--THEN BEGIN
|  |  |       +-------END
|  |  |       +--ELSE BEGIN
|  |  |       |       IF (MObjet = RangCourant)
|  |  |       |    +--THEN BEGIN
|  |  |       |    |       IF (BSomme^.NbSelect[RangCourant,Total] < 10) AND
|  |  |       |    |          (ToutAbaisse[RangCourant])
|  |  |       |    |       THEN Avertissement (ag_PourquoiDeselectSomme + Ord(MObjet))
|  |  |       |    +-------END
|  |  |       |       ELSE Avertissement (ag_TraiteRang + Ord(RangCourant));
|  |  |       +-------END;
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Recoit_MG_Ferme (IndOrig: Byte; VAR Msg: TMsgDidact);
   VAR Rang : VObjet;
       i : Byte;
+--BEGIN
|     WITH Calcul, Msg DO
|  +--IF IndOrig = 3 THEN BEGIN
|  |     IF MObjet = RangCourant
|  |  +--THEN BEGIN
|  |  |       ReportFait[RangCourant] := TRUE;
|  |  |       BCalcul^.PrendReport (Succ(RangCourant), 1);
|  |  |       IF ResultatOK[RangCourant] = 0
|  |  |    +--THEN BEGIN
|  |  |    |    +--IF IndicAction THEN BEGIN
|  |  |    |    |     Commentaire (CG_ReportFait + Ord(RangCourant), cg_Rien);
|  |  |    |    |     Avertissement (CG_YaPlusDe + Ord(RangCourant));
|  |  |    |    +--END;
|  |  |    |       BCalcul^.PrendResultat (RangCourant, 0);
|  |  |    |       DoneRang;
|  |  |    +-------END
|  |  |    +--ELSE BEGIN
|  |  |    |       IF IndicAction THEN Commentaire (CG_ReportFait + Ord(RangCourant),
|  |  |    |          CG_SelectReste + Ord(RangCourant));
|  |  |    +-------END;
|  |  +-------END
|  |  +--ELSE BEGIN
|  |  |       OuvertErreur[Succ(MObjet)] :=
|  |  |          (BSomme^.Card[Total][MObjet] > BSomme^.Card[Resultat][MObjet]);
|  |  |       IF NOT OuvertErreur[RangCourant]
|  |  |       THEN FOR i := 1 TO 2 DO WITH BNombre[i]^ DO OKSelect := OKSelect + [RangCourant];
|  |  |       IF (MGenre = NPos) THEN BCalcul^.PrendReport (Succ(MObjet), 1);
|  |  |       IF (MObjet < RangCourant) AND
|  |  |          (BSomme^.Card[Total][MObjet] = BSomme^.Card[Resultat][MObjet])
|  |  |       THEN BCalcul^.PrendResultat (MObjet, ResultatOK[MObjet]);
|  |  +-------END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Recoit_MG_Ouvre (IndOrig: Byte; VAR Msg: TMsgDidact);
   VAR i : Byte;
+--BEGIN
|  +--WITH Msg DO BEGIN
|  |     OuvertErreur [MObjet] := TRUE;
```

```
|  |     FOR i := 1 TO 2 DO
|  |        WITH BNombre[i]^ DO OKSelect := OKSelect - [RangCourant];
|  |
|  |     BCalcul^.PrendResultat (Pred(MObjet), ch_Interro);
|  |     IF MGenre = Report THEN BCalcul^.PrendReport (MObjet, ch_Espace);
|  |  +--CASE IndOrig OF
|  |  |  1, 2 : Avertissement (ag_OuvertureInutileNombre + Ord(MObjet));
|  |  |  3    : Avertissement (ag_OuvertureInutileSomme + Ord(MObjet));
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.RecoitMessage (VAR Msg: TMsgDidact);
   VAR IndOrig : Byte;
+--BEGIN
|     IF (Stop OR AddTerminee)
|     THEN Avertissement (cg_Valeurs)
|     ELSE
|  +--WITH Msg DO BEGIN
|  |     IF MOrigine = BNombre[1] THEN IndOrig := 1 ELSE
|  |     IF MOrigine = BNombre[2] THEN IndOrig := 2 ELSE
|  |     IF MOrigine = BSomme     THEN IndOrig := 3 ELSE
|  |     IF MOrigine = BCalcul    THEN IndOrig := 4 ELSE
|  |     IF MOrigine = BExplic    THEN IndOrig := 5;
|  |     IF MCode = rr_SansErreur
|  |  +--THEN CASE MMessage OF
|  |  |        MG_Select   : Recoit_MG_Select (IndOrig, Msg);
|  |  |        MG_Deselect : Recoit_MG_Deselect (IndOrig, Msg);
|  |  |        MG_Ferme    : Recoit_MG_Ferme (IndOrig, Msg);
|  |  |        MG_Ouvre    : Recoit_MG_Ouvre (IndOrig, Msg);
|  |  |        MG_Reduit   : BEGIN END;
|  |  +-------END
|  |  +--ELSE CASE MMessage OF
|  |  |        MG_Select   : Erreur_MG_Select (IndOrig, Msg);
|  |  |        MG_Deselect : Erreur_MG_Deselect (IndOrig, Msg);
|  |  |        MG_Ferme    : Erreur_MG_Ferme (IndOrig, Msg);
|  |  |        MG_Ouvre    : Erreur_MG_Ouvre (IndOrig, Msg);
|  |  |        MG_Reduit   : BEGIN END;
|  |  +-------END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Erreur_MG_Select (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|  +--WITH Msg DO BEGIN
|  |  +--CASE IndOrig OF
|  |  |  1, 2 : IF (MObjet <> Rien)
|  |  |       +--THEN BEGIN
|  |  |       |     +--CASE MCode OF
|  |  |       |     |  rr_MonoSelect : MessageBeep (mb_IconAsterisk);
|  |  |       |     |  rr_Genre : IF MObjet = RangCourant
|  |  |       |     |     THEN Avertissement (ag_NombresDejaCopies + Ord(RangCourant))
|  |  |       |     |     ELSE Avertissement (ag_RangDejaFait + Ord(MObjet));
|  |  |       |     |  rr_Objet :
|  |  |       |     |     IF (MObjet < RangCourant)
|  |  |       |     |     THEN Avertissement (ag_RangDejaFait + Ord(MObjet))
|  |  |       |     |     ELSE IF (MOBjet = RangCourant)
|  |  |       |     |          THEN Avertissement (ag_PasEncoreReferme + Ord(Ouverture))
|  |  |       |     |          ELSE Avertissement (ag_RangPasEncore + Ord(MObjet));
|  |  |       |     +--END;
|  |  |       +-------END
|  |  |       +--ELSE BEGIN
|  |  |       |       IF ToutAbaisse[RangCourant]
|  |  |       |       THEN Avertissement (ag_NombresDejaCopies + Ord(RangCourant))
|  |  |       |       ELSE Avertissement (ag_PasEncoreToutAbaisse + Ord(RangCourant));
|  |  |       +-------END;
|  |  |+-3    : BEGIN
|  |  ||         IF Ouverture <> Rien
|  |  ||      +--THEN BEGIN
|  |  ||      |       Avertissement (ag_PasEncoreReferme + Ord(Ouverture));
|  |  ||      +-------END
|  |  ||      +--ELSE BEGIN
|  |  ||      |       IF (MObjet = Rien)
|  |  ||      |    +--THEN BEGIN
|  |  ||      |    |       IF ToutAbaisse[RangCourant]
|  |  ||      |    |       THEN Avertissement (ag_TraiteRang + Ord(RangCourant))
```

```
|  |  ||     |    |       ELSE Avertissement (ag_PasEncoreToutAbaisse + Ord(RangCourant))
|  |  ||     |    +-------END
|  |  ||     |    +--ELSE BEGIN
|  |  ||     |    |       IF (MObjet < RangCourant)
|  |  ||     |    |       THEN Avertissement (ag_RangDejaFait + Ord(MObjet))
|  |  ||     |    |       ELSE IF (MObjet > RangCourant)
|  |  ||     |    |            THEN IF ToutAbaisse[RangCourant]
|  |  ||     |    |                 THEN Avertissement (ag_PasEncoreResultat + Ord(RangCourant)
|  |  ||     |    |                  )
|  |  ||     |    |                 ELSE Avertissement (ag_PasEncoreToutAbaisse + Ord(
|  |  ||     |    |                  RangCourant))
|  |  ||     |    +-------END;
|  |  ||     +-------END;
|  |  |+--------END;
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Erreur_MG_Deselect (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|  +--WITH Calcul, Msg DO BEGIN
|  |  +--CASE IndOrig OF
|  |  |  1, 2 : Avertissement (ag_PourquoiDeselectNombre + Ord(MObjet));
|  |  |  3    : IF Ouverture <> Rien
|  |  |      +--THEN BEGIN
|  |  |      |       IF MObjet = Ouverture
|  |  |      |       THEN Avertissement (ag_OKFermeture + Ord(Ouverture))
|  |  |      |       ELSE IF MObjet = RangCourant
|  |  |      |            THEN Avertissement (ag_PasEncoreReferme + Ord(Ouverture));
|  |  |      +-------END
|  |  |      +--ELSE BEGIN
|  |  |      |       IF ToutAbaisse[RangCourant] AND (MObjet = RangCourant)
|  |  |      |    +--THEN BEGIN
|  |  |      |    |       IF BSomme^.NbSelect[MObjet,Total] < Base
|  |  |      |    |       THEN Avertissement (ag_PourquoiDeselectSomme + Ord(MObjet));
|  |  |      |    +-------END;
|  |  |      +-------END;
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Erreur_MG_Ouvre (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|  +--WITH Msg DO BEGIN
|  |  +--CASE IndOrig OF
|  |  |  1, 2 : Avertissement (ag_RefusOuvertureNombre + Ord(MObjet));
|  |  |+-3    : BEGIN
|  |  ||       IF MObjet = Bonbon
|  |  ||       THEN Avertissement (ag_RefusOuvertureSomme + Ord(MObjet))
|  |  ||       ELSE IF (MObjet < RangCourant)
|  |  ||         +--THEN BEGIN
|  |  ||         |       IF MGenre = Resultat
|  |  ||         |       THEN Avertissement (ag_RangDejaFait + Ord(MObjet))
|  |  ||         |       ELSE Avertissement (ag_RefusOuvertureSomme + Ord(Ouverture));
|  |  ||         +-------END
|  |  ||            ELSE IF (MObjet >= RangCourant)
|  |  ||                 THEN Avertissement (ag_RefusOuvertureSomme + Ord(MObjet));
|  |  |+--------END;
|  |  +--END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd001.Erreur_MG_Ferme (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|  +--WITH Msg DO BEGIN
|  |     IF MCode = rr_SelectTrop THEN Avertissement (ag_TropPourFermer + Ord(MObjet));
|  +--END;
+--END;

   {-------------------------------------------------------------------------
            INITIALISATION
   -------------------------------------------------------------------------}
   END.
```

SAdd002.rc

```
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1000, "Nous allons additionner deux nombres en travaillant sur les chiffres au lieu d'utilise
les bonbons. Pour choisir des valeurs, clique sur le bouton \042Nouv. valeurs\042."
	1001, "Pour choisir deux valeurs, clique sur le bouton \042Nouv. valeurs\042. Eventuellement,
consulte le menu \042Options | Générales | Qui choisit les valeurs ?\042."
	1002, "Pour déplacer un chiffre de \042Somme des chiffres\042 vers \042Calcul écrit\042 : (1)
clique sur le chiffre et laisse le bouton enfoncé, (2) déplace le chiffre, (3) relâche le bouton de
la souris."
	1003, "Pour déplacer un chiffre de \042Somme des chiffres\042 vers \042Calcul écrit\042 : (1)
clique sur le chiffre et laisse le bouton enfoncé, (2) déplace le chiffre, (3) relâche le bouton de
la souris."
	1004, "L'addition est terminée. Pour choisir deux nouvelles valeurs, clique sur le bouton
\042Nouv. valeurs\042. Eventuellement, consulte le menu \042Options | Générales | Qui choisit les
valeurs ?\042."
	1005, "A gauche, tu as de 1 à 3 chiffres. A droite, tu as leur somme : un nombre entre 0 et
19. Les 1 ou 2 chiffres de cette somme, place-les comme résultat (et report) dans le calcul écrit."
	1006, "Quand tu as cliqué sur tous les chiffres au rang courant, enfonce le bouton
\042Calcule\042. Il te donnera un ou deux chiffres à replacer comme résultat (et report) dans le
calcul écrit."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1009, "Dans le calcul écrit, sélectionne tous les chiffres au rang des unités. Quand c'est
terminé, enfonce le bouton \042Calcule\042 pour avoir la somme de ces chiffres."
	1010, "Dans le calcul écrit, sélectionne tous les chiffres au rang des dizaines. Quand c'est
terminé, enfonce le bouton \042Calcule\042 pour avoir la somme de ces chiffres."
	1011, "Dans le calcul écrit, sélectionne tous les chiffres au rang des centaines. Quand c'est
terminé, enfonce le bouton \042Calcule\042 pour avoir la somme de ces chiffres."
	1012, "Dans le calcul écrit, sélectionne tous les chiffres au rang des milliers. Quand c'est
terminé, enfonce le bouton \042Calcule\042 pour avoir la somme de ces chiffres."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1025, "La somme des chiffres aux unités donne un nombre en 1 ou 2 chiffres : place-les au bor
endroit dans le calcul écrit."
	1026, "La somme des chiffres aux dizaines donne un nombre en 1 ou 2 chiffres : place-les au
bon endroit dans le calcul écrit."
	1027, "La somme des chiffres aux centaines donne un nombre en 1 ou 2 chiffres : place-les au
bon endroit dans le calcul écrit."
	1028, "La somme des chiffres aux milliers donne un nombre en 1 chiffre : place-le au bon
endroit dans le calcul écrit."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1041, "La somme des chiffres aux unités donne un nombre en 2 chiffres : le premier est un
report d'une dizaine, le deuxième est le résultat au rang des unités. Place les deux chiffres dans
calcul écrit."
	1042, "La somme des chiffres aux dizaines donne un nombre en 2 chiffres : le premier chiffre
est un report d'une centaine, le deuxième est le résultat au rang des dizaines. Place les deux
chiffres dans le calcul écrit."
	1043, "La somme des chiffres aux centaines donne un nombre en 2 chiffres : le premier chiffre
est un report d'un millier, le deuxième est le résultat au rang des centaines. Place les deux
chiffres dans le calcul écrit."
	1044, "Là, il y a un problème..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1057, "La somme des chiffres aux unités donne un nombre en 1 chiffre : place-le comme résulta
aux unités dans le calcul écrit."
	1058, "La somme des chiffres aux dizaines donne un nombre en 1 chiffre : place-le comme
résultat aux dizaines dans le calcul écrit."
	1059, "La somme des chiffres aux centaines donne un nombre en 1 chiffre : place-le comme
résultat aux centaines dans le calcul écrit."
	1060, "La somme des chiffres aux milliers donne un nombre en 1 chiffre : place-le comme
résultat aux milliers dans le calcul écrit."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1073, "Dans le calcul écrit, tu n'as pas sélectionné le chiffre des unités du premier nombre.
Il compte aussi dans le résultat aux unités."
	1074, "Tu oublies de sélectionner le chiffre des dizaines du premier nombre. Il compte aussi
dans le résultat aux dizaines."
```

```
	1075, "Dans le calcul écrit, le chiffre des centaines du premier nombre n'est pas encore
sélectionné."
	1076, "Tu n'as pas sélectionné le chiffre des milliers du premier nombre. Il intervient aussi
dans le calcul."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1089, "Tu oublies de sélectionner le chiffre des unités du second nombre."
	1090, "Tu oublies de sélectionner le chiffre des dizaines du second nombre. Il compte aussi
dans le résultat aux dizaines."
	1091, "Dans le calcul écrit, le chiffre des centaines du second nombre n'est pas encore
sélectionné."
	1092, "Tu n'as pas sélectionné le chiffre des milliers du second nombre. Il intervient aussi
dans le calcul."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1105, "Là, il y a un problème..."
	1106, "N'oublie pas : le report au rang des dizaines compte aussi dans le résultat.
Sélectionne-le..."
	1107, "Le report aux centaines fait changer le résultat : sélectionne aussi le chiffre du
report."
	1108, "Le report aux milliers fait changer le résultat : sélectionne aussi le chiffre du
report."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1121, "Nous travaillons sur des nombres entiers. Donc, il ne peut pas y avoir de report aux
unités."
	1122, "Il n'y a pas de report aux dizaines. Pas besoin de cliquer sur cette case."
	1123, "Il n'y a pas de report aux centaines. Pas besoin de cliquer sur cette case."
	1124, "Il n'y a pas de report aux milliers. Pas besoin de cliquer sur cette case."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1137, "Ce chiffre est déjà pris. Quand tu as sélectionné tous les chiffres des unités, clique
sur le bouton \042Calcule\042."
	1138, "Ce chiffre est déjà pris. Quand tu as sélectionné tous les chiffres des dizaines,
clique sur le bouton \042Calcule\042."
	1139, "Ce chiffre est déjà pris. Quand tu as sélectionné tous les chiffres des centaines,
clique sur le bouton \042Calcule\042."
	1140, "Ce chiffre est déjà pris. Quand tu as sélectionné tous les chiffres des milliers,
clique sur le bouton \042Calcule\042."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1153, "Là, il y a un problème..."
	1154, "Tu as déjà calculé le résultat pour ce rang. Passe aux dizaines."
	1155, "Tu as déjà calculé le résultat pour ce rang. Passe aux centaines."
	1156, "Tu as déjà calculé le résultat pour ce rang. Passe aux milliers."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1169, "Là, il y a un problème..."
	1170, "Suppose que tu calcules déjà le résultat aux dizaines. Si tu devais faire un report à
ce rang, tu devrais changer le résultat."
	1171, "Suppose que tu calcules déjà le résultat aux centaines. Si tu devais faire un report à
ce rang, tu devrais changer le résultat."
	1172, "Suppose que tu calcules déjà le résultat aux milliers. Si tu devais faire un report à
ce rang, tu devrais changer le résultat."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1185, "La somme des unités fait moins que 10. Il n'y a pas de report à faire aux dizaines."
	1186, "La somme des dizaines fait moins que 10. Il n'y a pas de report à faire aux centaines.
	1187, "La somme des centaines fait moins que 10. Il n'y a pas de report à faire aux milliers.
	1188, "Là, il y a un problème..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1201, "Ce 1 veut dire que tu transformes 10 bonbons en un sachet. Le sachet obtenu n'est pas
encore le résultat : il faut l'ajouter aux autres qui existent déjà. Place le 1 comme report au rang
des dizaines."
	1202, "Ce 1 veut dire que tu transformes 10 sachets en une caisse. La caisse obtenue n'est pas
encore le résultat : il faut l'ajouter aux autres qui existent déjà. Place le 1 comme report au rang
des centaines."
```

```
	1203, "Ce 1 veut dire que tu transformes 10 caisses en une armoire. L'armoire obtenue n'est pas encore le résultat : il faut l'ajouter aux autres qui existent déjà. Place le 1 comme report au rang des milliers."
	1204, "Là, il y a un problème..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1217, "Ce chiffre, c'est le nombre des bonbons qui restent après regroupement. Place-le comme résultat aux unités."
	1218, "Ce chiffre, c'est le nombre des sachets qui restent après regroupement. Place-le comme résultat aux dizaines."
	1219, "Ce chiffre, c'est le nombre des caisses qui restent après regroupement. Place-le comme résultat aux centaines."
	1220, "Ce chiffre, c'est le nombre d'armoires. Place-le comme résultat aux milliers."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1233, "Ce 1 veut dire que tu transformes 10 bonbons en un sachet : tu as 10 bonbons de moins, mais tu ajoutes un sachet. Place le 1 comme report au rang des dizaines."
	1234, "Ce 1 veut dire que tu transformes 10 sachets en une caisse : tu as 10 sachets de moins mais tu ajoutes une caisse. Place le 1 comme report au rang des centaines."
	1235, "Ce 1 veut dire que tu transformes 10 caisses en une armoire : tu as 10 caisses de moins, mais tu ajoutes une armoire. Place le 1 comme report au rang des milliers."
	1236, "Là, il y a un problème..."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1249, "Ce chiffre, c'est le nombre de bonbons qui restent après regroupement. Place-le comme résultat aux unités."
	1250, "Ce chiffre, c'est le nombre de sachets qui restent après regroupement. Place-le comme résultat aux dizaines."
	1251, "Ce chiffre, c'est le nombre de caisses qui restent après regroupement. Place-le comme résultat aux centaines."
	1252, "Ce chiffre, c'est le nombre d'armoires. Place-le comme résultat aux milliers."
END
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
	1265, "Place un chiffre comme résultat aux unités ou comme report aux dizaines."
	1266, "Place un chiffre comme résultat aux dizaines ou comme report aux centaines."
	1267, "Place un chiffre comme résultat aux centaines ou comme report aux milliers."
	1268, "Place ce chiffre comme résultat aux milliers."
END
```

GAdd002.pas

```
UNIT GAdd002;

{$R SAdd002.RC}

{-------------------------------------------------------------------------}
        INTERFACE
{-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, Strings,
     Globaux, DValeurs, ODidact, OCalcAS,
     OBoite, OBIcones, OBTexte, OBAddSub, OBouton, OBSimAdd;

TYPE
{-------------------------------------------------------------------------
     TGerantAdd002
-------------------------------------------------------------------------}

PGerantAdd002 = ^TGerantAdd002;
TGerantAdd002 = OBJECT (TGerantVide)
   BCalcul     : PBAddSub;
   BSimpleAdd  : PBSimpleAdd;
   BSomme      : PBIcones;
   BExplic     : PBTExte;
   BValeurs    : PBouton;
   BChiffres   : PBouton;

   Calcul      : TCalcAddSub;
   RangCourant : VObjet;
   NbChiffres  : Byte;
   RangTermine : Array[VObjet] of Boolean;
   ToutCopie   : Array[VObjet] of Boolean;
   ReportFait  : Array[VObjet] of Boolean;
   ADeplacer   : Set of VCategorie;
   Deplaces    : Set of VCategorie;
   Deplacement : Record
      Orig     : VCategorie;
      DestRang : VObjet;
      DestGenre: VCategorie;
      End;
   PushCalcule : Boolean;
   IndicAction : Boolean;
   AddTerminee : Boolean;

   constructor Init;
   destructor  Done;              virtual;

   procedure   ChangeParametres; virtual;
   procedure   ChoisitValeurs    (var Val1, Val2: Integer; var OK: Boolean);
   procedure   ClickBouton       (No: Byte); virtual;
   procedure   Commentaire       (ID1, ID2 : Integer);
   procedure   DoneExercice;     virtual;
   procedure   DoneRang;
   procedure   ErreurGenerale;
   procedure   Erreur_MG_Deselect(IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Erreur_MG_Prend   (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Erreur_MG_Select  (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   GarnitMainWindow; virtual;
   procedure   InitExercice      (Param: Pointer); virtual;
   procedure   InitRang;
   procedure   RecoitMessage     (var Msg: TMsgDidact); virtual;
   procedure   Recoit_MG_Depose  (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Recoit_MG_Prend   (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   Recoit_MG_Select  (IndOrig: Byte; var Msg: TMsgDidact);
   procedure   VerifieChiffresCE;
   END;

{-------------------------------------------------------------------------}
        IMPLEMENTATION
{-------------------------------------------------------------------------}

CONST
{-------------------------------------------------------------------------
     Identificateurs des chaînes de caractères
```

```
            cg_XXXX : instructions pour effectuer l'exercice
            ag_XXXX : messages lors d'une mauvaise manipulation
   ------------------------------------------------------------------------------}
   cg_Rien              = -0001;
   cg_Inchange          = -0002;

   cg_Debut             = 1000;
   cg_ChoisisDesValeurs= 1001;
   cg_CommentDeplacer   = 1002;
   cg_Fin               = 1004;
   cg_SelectCETout      = 1008;
   cg_SelectCEn12       = 1900;
   cg_SelectCEn12R      = 1950;
   cg_CopieASTout       = 1025;
   cg_CopieASResRep     = 1040;
   cg_CopieASResultat   = 1056;


   ag_Chiffre1PasSelect       = 1072;
   ag_Chiffre2PasSelect       = 1088;
   ag_ReportPasSelect         = 1104;
   ag_PasDeReportCE           = 1120;
   ag_ChiffresCEDejaCopies    = 1136;
   ag_RangDejaFait            = 1152;
   ag_RangPasEncore           = 1168;
   ag_PasDeReportSA           = 1184;
   ag_CaseInvalideSA          = 1005;
   ag_ReportVersResultat      = 1200;
   ag_ResultatVersReport      = 1216;
   ag_ReportMauvaisRang       = 1232;
   ag_ResultatMauvaisRang     = 1248;
   ag_VersMauvaiseCase        = 1264;
   ag_EnfonceBoutonCalcule    = 1006;

   {------------------------------------------------------------------------------
       TGerantAdd002
   ------------------------------------------------------------------------------}

   CONSTRUCTOR TGerantAdd002.Init;
   BEGIN
      TGerantVide.Init;
      Fenetre^.SetCaption ('Addition en chiffres');
      AddTerminee := TRUE;
      Commentaire (CG_Debut, CG_Rien);
   END;
   {------------------------------------------------------------------------------}
   DESTRUCTOR TGerantAdd002.Done;
+--BEGIN
|     VideMainWindow;
+--END;
   {------------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.ChangeParametres;
+--BEGIN
|  +--WITH Fenetre^.Parametres DO BEGIN
|  |     IndicAction := Guide = gd_Detaille;
|  |     BSimpleAdd^.QuiCalcule := Tables;
|  +--END;
+--END;
   {------------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.ClickBouton (No: Byte);
+--BEGIN
|  +--CASE No OF
|  |  1 : InitExercice (NIL);
|  |  2 : IF AddTerminee
|  |       THEN Avertissement (cg_ChoisisDesValeurs)
|  |    +--ELSE BEGIN
|  |    |       PushCalcule := TRUE;
|  |    |       VerifieChiffresCE;
|  |    |       PushCalcule := FALSE;
|  |    +-------END;
|  +--END;
+--END;
   {------------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.ChoisitValeurs (VAR Val1, Val2: INTEGER; VAR OK: BOOLEAN);
   VAR But : INTEGER;
+--BEGIN
```

```
|  +--CASE Fenetre^.Parametres.Choix OF
|  |+-ch_Eleve : BEGIN
|  ||    OK := DialogueValeurs (Fenetre, Val1, '+', Val2);
|  |+----END;
|  |+-ch_Professeur : BEGIN
|  ||    Randomize;
|  || +--CASE Fenetre^.Parametres.NBRangs OF
|  || |  3 : But := 2 + Random (998);
|  || |  4 : But := 2 + Random(9998);
|  || +--END;
|  ||    Val1 := 1 + Random(But);
|  ||    Val2 := But - Val1;
|  ||    OK := TRUE;
|  |+----END;
|  |+-ch_Ordinateur : BEGIN
|  ||    Randomize;
|  || +--CASE Fenetre^.Parametres.NBRangs OF
|  || |  3 : But := 2 + Random (998);
|  || |  4 : But := 2 + Random(9998);
|  || +--END;
|  ||    Val1 := 1 + Random(But);
|  ||    Val2 := But - Val1;
|  ||    OK := TRUE;
|  |+----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.Commentaire (ID1, ID2 : INTEGER);
   VAR Buffer1 : TCommentaire;
       Buffer2 : TCommentaire;
       NL      : ARRAY[1..3] OF CHAR;
+--BEGIN
|     MetAZero (Buffer1, SizeOf (Buffer1));
|     MetAZero (Buffer2, SizeOf (Buffer2));
|     IF ID1 <> cg_Rien THEN LoadString (HInstance, ID1, @Buffer1, MaxComment);
|     IF ID2 <> cg_Rien THEN LoadString (HInstance, ID2, @Buffer2, MaxComment);
|     NL := #$0D#$0A#$00;
|     StrLCat (@Buffer1, @NL, MaxComment);
|     StrLCat (@Buffer1, @Buffer2, MaxComment);
|     BExplic^.PrendTexte (Buffer1);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.DoneExercice;
+--BEGIN
|     Commentaire (CG_Fin, CG_Rien);
|     AddTerminee := TRUE;
|     Stop := TRUE;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.DoneRang;
+--BEGIN
|     BSomme^.VerrouilleObjet (RangCourant);
|     BCalcul^.SelectionneRang (RangCourant, FALSE);
|     BCalcul^.OKSelect := [];
|     BSimpleAdd^.VideContenu;
|     IF RangCourant = Calcul.RangMax
|  +--THEN BEGIN
|  |       DoneExercice;
|  +-------END
|  +--ELSE BEGIN
|  |       Inc (RangCourant);
|  |       InitRang;
|  +-------END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.ErreurGenerale;
+--BEGIN
|     IF ToutCopie[RangCourant]
|  +--THEN BEGIN
|  |       IF BSimpleAdd^.SommeDonnee
|  |    +--THEN BEGIN
|  |    |       IF Calcul.ReportOKDe[RangCourant]
|  |    |       THEN Avertissement (cg_CopieASResRep + Ord(RangCourant))
|  |    |       ELSE Avertissement (cg_CopieASResultat+Ord(RangCourant));
|  |    +-------END
|  |       ELSE Avertissement (ag_EnfonceBoutonCalcule);
```

```
|  +-------END
|     ELSE VerifieChiffresCE;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.Erreur_MG_Deselect (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|     WITH Msg DO
|  +--CASE IndOrig OF
|  |+-1 : BEGIN
|  ||     ErreurGenerale;
|  |+-----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.Erreur_MG_Prend (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|     WITH Msg DO
|  +--CASE IndOrig OF
|  |  2 : IF ToutCopie[RangCourant]
|  |   +--THEN BEGIN
|  |   |       IF BSimpleAdd^.Contenu[7] = ''
|  |   |       THEN Avertissement (ag_EnfonceBoutonCalcule)
|  |   |    +--ELSE CASE MNoCase OF
|  |   |    |       0..5 : IF Calcul.ReportOKDe[RangCourant]
|  |   |    |              THEN Avertissement (cg_CopieASResRep +
|  |   |    |                   Ord(RangCourant))
|  |   |    |              ELSE Avertissement (cg_CopieASResultat+
|  |   |    |                   Ord(RangCourant));
|  |   |    |       6    : Avertissement (ag_PasDeReportSA+Ord(RangCourant));
|  |   |    +-------END
|  |   +-------END
|  |      ELSE VerifieChiffresCE;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.Erreur_MG_Select (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|     WITH Msg DO
|  +--CASE IndOrig OF
|  |+-1: BEGIN
|  || +--IF MObjet = Rien THEN BEGIN
|  || |     ErreurGenerale;
|  || |     Exit;
|  || +--END;
|  || +--IF MObjet < RangCourant THEN BEGIN
|  || |     Avertissement (ag_RangDejaFait+Ord(RangCourant));
|  || |     Exit;
|  || +--END;
|  || +--IF MObjet > RangCourant THEN BEGIN
|  || |     Avertissement (ag_RangPasEncore+Ord(MObjet));
|  || |     Exit;
|  || +--END;
|  || +--CASE MGenre OF
|  || |  Report : BEGIN END;
|  || |  NPos   : BEGIN END;
|  || |  NNeg   : BEGIN END;
|  || |  ELSE ErreurGenerale;
|  || +--END;
|  |+----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.GarnitMainWindow;
   VAR Max     : TPoint;
       DimBIc2 : TPoint;
       BordFen : TPoint;
       X, Y    : ARRAY[1..9] OF INTEGER;
       Texte   : ARRAY[0..20] OF CHAR;
+--BEGIN
|     Max.X := GetSystemMetrics (sm_cxFullScreen);
|     Max.Y := GetSystemMetrics (sm_cyFullScreen) - GetSystemMetrics (sm_cyMenu);
|     DimBIc2.X := 2*(BordIntBoite + 1) + 19*(LargeurIcone + EntreIcones);
|     DimBIc2.Y := 2*(HauteurTitre + BordIntBoite + 1) +
|        4*(HauteurIcone + EntreIcones);
|     BordFen.X := (Max.X - DimBIc2.X) DIV 2;
|     BordFen.Y := (Max.Y - 3*DimBIc2.Y) DIV 2;
```

```
|
|     X[1] := BordFen.X;
|     Y[1] := BordFen.Y;
|     X[2] := Max.X - BordFen.X;
|     Y[2] := Y[1] + (3*DimBIc2.Y) DIV 4;
|     BExplic := New (PBTexte,
|        Init(Fenetre,'Commentaires',X[1],Y[1],X[2],Y[2]));
|     Fenetre^.AjouteBoite (BExplic);
|
|     X[3] := BordFen.X;
|     Y[3] := Max.Y - BordFen.Y - DimBIc2.Y;
|     BSomme := New (PBIcones, Init(Fenetre,'Addition',X[3],Y[3],19,4));
|     Fenetre^.AjouteBoite (BSomme);
|
|     StrLCopy (@Texte, 'Nouv. valeurs', 20);
|     TailleBouton (Fenetre, @Texte, X[7], Y[7]);
|
|     X[4] := Max.X DIV 2 + BordFen.X DIV 2;
|     Y[4] := Y[2] + Y[7] + 2*10;
|     BCalcul := New (PBAddSub, Init(Fenetre,'Calcul écrit',X[4],Y[4]));
|     Fenetre^.AjouteBoite (BCalcul);
|
|     X[5] := Max.X DIV 2 - BordFen.X DIV 2 - 2*BordIntBoite - 7*20;
|     Y[5] := Y[4];
|     BSimpleAdd := New (PBSimpleAdd,
|        Init (Fenetre, 'Somme des chiffres', X[5], Y[5]));
|     Fenetre^.AjouteBoite (BSimpleAdd);
|
|     X[6] := X[4] + (BCalcul^.X2 - X[4] - X[7]) DIV 2;
|     Y[6] := Y[2] + 10;
|     Fenetre^.AjouteBouton (BValeurs,Texte, X[6],Y[6],X[6]+X[7],Y[6]+Y[7]);
|
|     StrLCopy (@Texte, 'Calcule', 20);
|     TailleBouton (Fenetre, @Texte, X[9], Y[9]);
|     X[8] := X[5] + (BSimpleAdd^.X2 - X[5] - X[{9}7]) DIV 2;
|     Y[8] := Y[6];
|     Fenetre^.AjouteBouton (BChiffres,Texte, X[8],Y[8],X[8]+X[{9}7],Y[8]+Y[{9}7]);
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.InitExercice (Param: Pointer);
   VAR Val  : ARRAY[1..2] OF INTEGER;
       Rang : VObjet;
       I    : INTEGER;
       OK   : BOOLEAN;
+--BEGIN
|     Calcul.Base := 10;
|     IF Param = NIL
|     THEN ChoisitValeurs (Val[1], Val[2], OK)
|  +--ELSE BEGIN
|  |       Move (Param^, Val, 4);
|  |       Val[1] := Abs(Val[1]);
|  |       Val[2] := Abs(Val[2]);
|  |       OK := Val[1] + Val[2] <= 9999;
|  +-------END;
|
|  +--IF OK THEN BEGIN
|  |  +--WITH BSomme^ DO BEGIN
|  |  |     MonoSelect := TRUE;
|  |  |     OKOuvrir   := [];
|  |  |     OKFermer   := [];
|  |  |     OKSelect   := [];
|  |  |     OKDeselect := [];
|  |  |     PrendValeur (0, Base);
|  |  |     Usage (ub_Statique);
|  |  +--END;
|  |     Calcul.PrendValeurs (Val[1], '+', Val[2], Calcul.Base);
|  |     BCalcul^.PrendValeurs (Val[1], '+', Val[2], Calcul.Base);
|  |     BCalcul^.OKSelect := [];
|  |     BCalcul^.OKDeselect := [];
|  |     BSimpleAdd^.VideContenu;
|  |     IndicAction := PFenetreVide(Application^.MainWindow)^.Parametres.Guide = gd_Detaille;
|  |     MetAZero (RangTermine, SizeOf (RangTermine));
|  |     MetAZero (ToutCopie, SizeOf (ToutCopie));
|  |     RangCourant := Bonbon;
|  |     AddTerminee := FALSE;
|  |     Stop        := FALSE;
```

```
|  |     PushCalcule := FALSE;
|  |     InitRang;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.InitRang;
+--BEGIN
|     BCalcul^.AccepteRang (RangCourant, TRUE);
|     BSimpleAdd^.OKCapture := [];
|     Deplaces := [];
|     IF Calcul.ReportOKDe[RangCourant]
|     THEN ADeplacer := [Report, Resultat]
|     ELSE ADeplacer := [Resultat];
|     Commentaire (cg_SelectCETout+Ord(RangCourant), cg_Rien);
|  +--WITH Deplacement DO BEGIN
|  |     Orig := Total;
|  |     DestRang := Rien;
|  |     DestGenre := Total;
|  +--END;
|     NbChiffres := 0;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.RecoitMessage (VAR Msg: TMsgDidact);
   VAR IndOrig : Byte;
+--BEGIN
|     IF (Stop OR AddTerminee)
|     THEN Avertissement (cg_ChoisisDesValeurs)
|     ELSE
|  +--WITH Msg DO BEGIN
|  |     IF MOrigine = BCalcul    THEN IndOrig := 1 ELSE
|  |     IF MOrigine = BSimpleAdd THEN IndOrig := 2 ELSE
|  |     IF MOrigine = BSomme     THEN IndOrig := 3 ELSE
|  |     IF MOrigine = BExplic    THEN IndOrig := 4;
|  |     IF MCode = rr_SansErreur
|  |  +--THEN CASE MMessage OF
|  |  |        MG_Select   : Recoit_MG_Select   (IndOrig, Msg);
|  |  |        MG_Prend    : Recoit_MG_Prend    (IndOrig, Msg);
|  |  |        MG_Depose   : Recoit_MG_Depose   (IndOrig, Msg);
|  |  +-------END
|  |  +--ELSE CASE MMessage OF
|  |  |        MG_Select   : Erreur_MG_Select   (IndOrig, Msg);
|  |  |        MG_Deselect : Erreur_MG_Deselect (IndOrig, Msg);
|  |  |        MG_Prend    : Erreur_MG_Prend    (IndOrig, Msg);
|  |  +-------END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.Recoit_MG_Depose (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|     WITH Msg DO
|  +--CASE IndOrig OF
|  |+-1: WITH Deplacement DO BEGIN
|  ||       DestRang := MObjet;
|  ||       DestGenre := MGenre;
|  |+----END;
|  |+-2: WITH Deplacement DO BEGIN
|  ||    IF Orig = Resultat
|  ||    THEN BSomme^.SelectionneObjet (RangCourant, FALSE)
|  ||    ELSE BSomme^.SelectionneObjet (Succ(RangCourant), FALSE);
|  || +--CASE Orig OF
|  || |  Report  :
|  || |     IF DestRang = Succ(RangCourant)
|  || |  +--THEN CASE DestGenre OF
|  || |  |     +--Report  : BEGIN
|  || |  |     |     Deplaces := Deplaces + [Report];
|  || |  |     |  +--WITH Deplacement DO BEGIN
|  || |  |     |  |     Orig := Total;
|  || |  |     |  |     DestRang := Rien;
|  || |  |     |  |     DestGenre := Total;
|  || |  |     |  +--END;
|  || |  |     |     BCalcul^.PrendReport (Succ(RangCourant), 1);
|  || |  |     +-----END;
|  || |  |        Resultat: Avertissement (ag_ReportVersResultat+Ord(RangCourant));
|  || |  |        ELSE Avertissement (ag_VersMauvaiseCase+Ord(RangCourant));
|  || |  +-------END
|  || |     ELSE Avertissement (ag_ReportMauvaisRang + Ord(RangCourant));
```

```
|  || |  Resultat:
|  || |     IF DestRang = RangCourant
|  || |  +--THEN CASE DestGenre OF
|  || |  |     +--Resultat: BEGIN
|  || |  |     |     Deplaces := Deplaces + [Resultat];
|  || |  |     |  +--WITH Deplacement DO BEGIN
|  || |  |     |  |     Orig := Total;
|  || |  |     |  |     DestRang := Rien;
|  || |  |     |  |     DestGenre := Total;
|  || |  |     |  +--END;
|  || |  |     |     BCalcul^.PrendResultat
|  || |  |     |        (RangCourant, Calcul.ResultatOK[RangCourant]);
|  || |  |     +-----END;
|  || |  |        Report  : Avertissement (ag_ResultatVersReport +
|  || |  |            Ord(RangCourant));
|  || |  |        ELSE Avertissement (ag_VersMauvaiseCase+ord(RangCourant));
|  || |  +-------END
|  || |     ELSE Avertissement (ag_ResultatMauvaisRang + Ord(RangCourant));
|  || +--END;
|  ||    IF Deplaces = ADeplacer THEN DoneRang;
|  |+----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.Recoit_MG_Prend (IndOrig: Byte; VAR Msg: TMsgDidact);
+--BEGIN
|     WITH Msg DO
|  +--CASE IndOrig OF
|  |+-2 : BEGIN
|  ||  +--CASE MGenre OF
|  ||  |+-Report   : BEGIN
|  ||  ||    Deplacement.Orig := Report;
|  ||  ||    BSomme^.SelectionneObjet (Succ(RangCourant), TRUE);
|  ||  |+----END;
|  ||  |+-Resultat : BEGIN
|  ||  ||    Deplacement.Orig := Resultat;
|  ||  ||    BSomme^.SelectionneObjet (RangCourant, TRUE);
|  ||  |+----END;
|  ||  +--END;
|  |+-----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.Recoit_MG_Select (IndOrig: Byte; VAR Msg: TMsgDidact);
   VAR Chiffre : Byte;
       NouvCard: TCardinal;
+--BEGIN
|     WITH Msg DO
|  +--CASE IndOrig OF
|  |+-1: BEGIN
|  ||    Inc(NbChiffres);
|  || +--CASE MGenre OF
|  || |  NPos  : Chiffre := Calcul.Nombre[1,RangCourant];
|  || |  NNeg  : Chiffre := Calcul.Nombre[2,RangCourant];
|  || |  Report: Chiffre := 1;
|  || +--END;
|  ||    IF MGenre = Report
|  ||    THEN BSimpleAdd^.AjouteValeur (Report, Chiffre)
|  ||    ELSE BSimpleAdd^.AjouteValeur (NPos, Chiffre);
|  ||    NouvCard := BSomme^.Card;
|  ||    IF MGenre <> Report THEN Inc (NouvCard[NPos, RangCourant], Chiffre);
|  ||    BSomme^.PrendCardinal (NouvCard, Calcul.Base);
|  ||    ToutCopie[RangCourant] := BCalcul^.CaseSelect = BCalcul^.OKSelect;
|  |+----END;
|  +--END;
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TGerantAdd002.VerifieChiffresCE;
   VAR ResteCases : SET OF 0..24;
       ResteGenres: SET OF VCategorie;
       NoCase     : Byte;
       NouvCard   : TCardinal;
       Id2        : INTEGER;
       UnObjet    : VObjet;
+--BEGIN
|     ResteCases := BCalcul^.OKSelect - BCalcul^.CaseSelect;
```

```
|     ResteGenres := [];
|  +--IF ResteCases <> [] THEN BEGIN
|  |     FOR NoCase := 0 TO 24 DO
|  |      +--IF NoCase IN ResteCases THEN BEGIN
|  |      |     IF (NoCase DIV 5) = 0
|  |      |     THEN ResteGenres := ResteGenres + [Report];
|  |      |     IF ((NoCase DIV 5) = 1) AND (Calcul.Nombre[1,RangCourant] <> 0)
|  |      |     THEN ResteGenres := ResteGenres + [NPos];
|  |      |     IF ((NoCase DIV 5) = 2) AND (Calcul.Nombre[2,RangCourant] <> 0)
|  |      |     THEN ResteGenres := ResteGenres + [NNeg];
|  |      +--END;
|  |     IF Report IN ResteGenres
|  |     THEN Avertissement (ag_ReportPasSelect+Ord(RangCourant)) ELSE
|  |     IF NPos IN ResteGenres
|  |     THEN Avertissement (ag_Chiffre1PasSelect+Ord(RangCourant)) ELSE
|  |     IF NNeg IN ResteGenres
|  |     THEN Avertissement (ag_Chiffre2PasSelect+Ord(RangCourant));
|  +--END;
|  +--IF ResteGenres = [] THEN BEGIN
|  |     IF PushCalcule
|  |  +--THEN BEGIN
|  |  |        BSimpleAdd^.CalculeSomme;
|  |  |        NouvCard := BSomme^.Card;
|  |  |        NouvCard[NPos,RangCourant] := Calcul.ResultatOK[RangCourant];
|  |  |        NouvCard[Report,RangCourant] := 0;
|  |  |        IF Calcul.ReportOKDe[RangCourant]
|  |  |        THEN NouvCard[Report,Succ(RangCourant)] := 1
|  |  |        ELSE NouvCard[Report,Succ(RangCourant)] := 0;
|  |  |        BSomme^.PrendCardinal (NouvCard, Calcul.Base);
|  |  |        IF IndicAction
|  |  |     +--THEN BEGIN
|  |  |     |        Id2 := cg_CommentDeplacer;
|  |  |     |        FOR UnObjet := Bonbon TO Pred(RangCourant) DO
|  |  |     |            IF Calcul.ReportOKDe[UnObjet] THEN Id2 := cg_Rien;
|  |  |     |        IF Calcul.ReportOKDe[RangCourant]
|  |  |     |        THEN Commentaire (cg_CopieASResRep+Ord(RangCourant), Id2)
|  |  |     |        ELSE Commentaire (cg_CopieASResultat+Ord(RangCourant), Id2);
|  |  |     +-------END
|  |  |        ELSE Commentaire (cg_CopieASTout+Ord(RangCourant), cg_Rien);
|  |  +-------END
|  |     ELSE Avertissement (ag_EnfonceBoutonCalcule);
|  +--END;
+--END;

  {-------------------------------------------------------------------------
           INITIALISATION
  -------------------------------------------------------------------------}
  END.
```

Addition.pas

```
PROGRAM Addition;

{-------------------------------------------------------------------------
         INTERFACE
-------------------------------------------------------------------------}

USES OWindows, WinTypes, WinProcs, WinDos, Strings,
     OBTexte, ODidact, GAdd001, GAdd002;

CONST
cm_Ex1 = 111;
cm_Ex2 = 112;

TYPE
{-------------------------------------------------------------------------
     Pointeurs
-------------------------------------------------------------------------}
PDidacticiel     = ^TDidacticiel;
PFenetreDidact   = ^TFenetreDidact;

{-------------------------------------------------------------------------
    TDidacticiel : application Windows qui utilise les objets didactiques
-------------------------------------------------------------------------}

TDidacticiel = OBJECT (TDidactVide)
   procedure   InitMainWindow; virtual;
   END;

{-------------------------------------------------------------------------
    TFenetreDidact : fenêtre (principale) du didacticiel
-------------------------------------------------------------------------}

TFenetreDidact = OBJECT (TFenetreVide)
   constructor Init (UnParent: PWindowsObject; UnTitre: PChar);
   procedure   SetupWindow; virtual;
   procedure   LanceExercice1 (var Msg:TMessage); virtual cm_First + cm_Ex1;
   procedure   LanceExercice2 (var Msg:TMessage); virtual cm_First + cm_Ex2;
   END;

{-------------------------------------------------------------------------
         IMPLEMENTATION
-------------------------------------------------------------------------}

VAR Prg : TDidacticiel;

{-------------------------------------------------------------------------
         TDidacticiel
-------------------------------------------------------------------------}

+--PROCEDURE TDidacticiel.InitMainWindow;
+--BEGIN
|     MainWindow := New (PFenetreDidact, Init (NIL, 'Addition'));
+--END;

{-------------------------------------------------------------------------
         TFenetreDidact
-------------------------------------------------------------------------}

CONSTRUCTOR TFenetreDidact.Init (UnParent: PWindowsObject; UnTitre: PChar);
BEGIN
   TFenetreVide.Init (UnParent, UnTitre);
END;
{-------------------------------------------------------------------------}
PROCEDURE TFenetreDidact.LanceExercice1 (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Dispose (Gerant, Done);
|     Gerant := New (PGerantAdd001, Init);
+--END;
{-------------------------------------------------------------------------}
PROCEDURE TFenetreDidact.LanceExercice2 (VAR Msg: TMessage);
+--BEGIN
|     IF Gerant <> NIL THEN Dispose (Gerant, Done);
|     Gerant := New (PGerantAdd002, Init);
```

```
+--END;
   {-------------------------------------------------------------------------}
   PROCEDURE TFenetreDidact.SetupWindow;
   VAR Msg: TMessage;
       IDMenu: HMenu;
+--BEGIN
|     LanceExercice1 (Msg);
|     TFenetreVide.SetupWindow;
|     IDMenu := GetMenu(Application^.MainWindow^.HWindow);
|     InsertMenu (IDMenu, cm_Debut, mf_ByCommand, cm_Ex1, PChar('Exercice 1'));
|     InsertMenu (IDMenu, cm_Debut, mf_ByCommand, cm_Ex2, PChar('Exercice 2'));
|     InsertMenu (IDMenu, cm_Debut, mf_ByCommand OR mf_Separator, 113, NIL);
+--END;

   {-------------------------------------------------------------------------
            INITIALISATION
   -------------------------------------------------------------------------}
+--BEGIN
|  Prg.Init('Mémoire','Graduate');
|  Prg.Run;
|  Prg.Done;
+--END.
```